The PNC 2 Cluster Algorithm

An integrated learning algorithm for rule induction

Lars Haendel (lars@haendels.de)
Dortmund, Germany 2003

This document is an english translation of the parts of my PhD thesis, that are relevant to the PNC 2 cluster algorithm:

Lars Haendel. *Clusterverfahren zur datenbasierten Generierung interpretierbarer Regeln unter Verwendung lokaler Entscheidungskriterien.* PhD thesis, University of Dortmund, Faculty of Electrical Engineering and Information Technologies, 2003

Abstract This document describes the hierarchical agglomerative cluster algorithm PNC 2 in the context of direct generation of IF-THEN rules for classification tasks. As an agglomerative cluster algorithm, the PNC 2 initializes each learn data tuple as a single cluster. Then, if a merge test is passed, iteratively always those two clusters with the same output value are merged, that are closest to each other. The merge test transforms the generalized cluster into a rule and evaluates it by a kind of hitrate. The rule's premise is the cuboid, that encloses the input vectors of all learn data tuples merged in the cluster. This representation suffers in high dimensional input spaces due to the COD problem and thus a special mechanism is used to extend the cuboid during the merge test.

A heterogenous normalized and weighted Minkowski overlap metric is used to be able to process mixed continuous and nominal inputs. An integrated bagging component can improve accuracy and also reduces the time complexity for a learn data sample with N data tuples from $O(N^3)$ to approximately $O(N^2)$. The size of the learned rule set can be further reduced by applying a context sensitive feature selection, that individually removes the unnecessary inputs from each rule's premise. The algorithm can also be viewed as an instance based learning algorithm, namely as an exemplar-based generalization approach. Thus the idea of the k-nearest-neighbor algorithm (kNN), to base the decision on several surrounding learn data tuples, can be transferred to improve the prediction accuracy.

The number of free parameters of the PNC 2 is been reduced in a preliminary study with some development benchmarks. Then the PNC 2 is compared experimentally with the most similar existing algorithms, namely with the NGE, the RISE and, of course, with the kNN algorithm. All remaining free parameters of the PNC 2 are tuned using cross-validation or similar approaches within the respective learn data samples. The PNC 2 outperforms the NGE algorithms and its variants and reaches better or comparative accuracies as the kNN or the RISE algorithm - with typically much smaller ruleset/model sizes.

Acknowledgement The PNC 2 cluster algorithm was developed while I was a scholarship holder in the post graduate research program *Modelling and Model-Based Design of Complex Technological Systems* at the *Chair of Electrical Control Engineering* at the University of Dortmund, Germany. My research project was initiated by Prof. Dr. rer. nat. H. Kiendl. The post graduate research program was founded by the *Deutsche Forschungsgemeinschaft* (DFG).

Note The PNC 2 RULE INDUCTION SYSTEM is a free Windows software tool, that is using the PNC 2 cluster algorithm to automatically induce rules from a given data sample. Additionally a DOS command line version will be available soon. The kernels are written in ANSI C++, they are well documented and should easily be compiled for different operating systems. You may download the program at `http://www.newty.de/pnc2/index.html`.

Contents

Preface

Introductory Remarks

Models for the input/output behavior of a system are of great interest in everyday or technical life. They help to get an insight into a system or they can be used to predict the system's output for a particular input position. A model is always an abstraction, that corresponds to the reality only with respect to some pre-specified aspects.

One possible approach to model a given system is to set up and solve the equations, that describe the underlying physical effects of the system. Unknown parameters can then be identified in experiments with the real system. The resulting models are highly transparent and interpretable. They allow an analytical analysis, and maybe they also allow conclusions about the system's behavior for a continuum of possible parameter values or operating conditions. This approach is also called *deductive* or *white-box* modelling. A disadvantage of this approach is, that a deep understanding of the system is necessary and/or the modelling requires a lot of work and time.

Often data-based modelling is a suitable alternative. First some data, that should reveal all relevant correlations or operating conditions, is collected from the real system. Then, based on this data, a model is learned using a suitable learning algorithm. None, or only little, knowledge about the system is necessary. This approach is also called *inductive* or *grey-* resp. *black-box* modelling. The classification as black- or grey-box modelling is subjective and reflects, how easy the learned model can be interpreted by a human.

One possible approach for data-based modelling is the learning of If-Then rules. Each rule describes a relevant part of the system for a particular part of the input space. Thus rules are an intuitively understandable way to represent knowledge about a system. It is further possible to directly integrate and use existing human knowledge within a single framework.

Induction is only one kind of machine learning. Very similar is *data-mining*, where one tries to find all valid or interesting rules in large and complex databases using mostly statistical methods to perform a kind of explorative data analysis.

Structure of this Work

First of all, chapter 1 introduces the task of data-based modelling and describes the typical process of how to learn a model. Afterwards the basics of rule-based models and of instance-based learning are explained. Then a quick overview about the problem of the *curse of dimensionality* and about feature selection methods is given and the two basic paradigms of clustering algorithms are elucidated. Following, the Minkowski metric is introduced and various methods of how to normalize and weight the input features are discussed. The chapter is finished with a short summary of how cluster algorithms can be used in the context of data-based modelling and the denomination of the goals of this work.

Chapter 2 describes the *Positive and Negative example-based Clustering* (Pnc 2) algorithm. First the basic idea is outlined. Then the simplified version of the algorithm is described and the pseudo-code is presented. Afterwards advanced details and enhancements are given and the most similar existing algorithms are identified and compared with the Pnc 2 cluster algorithm.

Chapter 3 first gives an introduction to different loss functions to evaluate the prediction accuracy of a learned model or of a learning algorithm itself. Two methods – namely the N-fold cross-validation and the N-fold repetition – of how to estimate the loss function value of a learning algorithm for a given data sample are described. A problem arises, if the algorithm has free tunable parameters. These parameters must be adjusted systematically – otherwise the results can be corrupted. Thus, based on a work of Salzberg, the approach of the *tough validation* is defined.

Chapter 4 first describes the preliminary thoughts and experiments done to get a deep understanding about the algorithm, to reduce the number of free parameters, to find some good standard sets for the remaining parameters and to decide upon the strategies used for the parameter tuning. Afterwards the PNC 2 cluster algorithm is compared with the most similar existing approaches in an extensive benchmark study.

Chapter 5 finally summarizes this work and gives an outlook about future work.

Take a look at the tables D.2 and D.3 in appendix D to get familiar with some special counters, indices and terms used throughout this document.

Chapter 1

The Basics

First of all, chapter 1 introduces the task of data-based modelling and describes the typical process of how to learn a model. Afterwards the basics of rule-based models and of instance-based learning are explained. Then a quick overview about the problem of the *curse of dimensionality* and about feature selection methods is given and the two basic paradigms of clustering algorithms are elucidated. Following the Minkowski metric is introduced and various methods of how to normalize and weight the input features are discussed. The chapter is finished with a short summary of how cluster algorithms can be used in the context of data-based modelling and the denomination of the goals of this work.

1.1 Data-Based Modelling

The Basic Task Figure 1.1 illustrates the basic task of data-based modelling. Given is a system with an input vector $\mathbf{x}$ and the corresponding output y. The input vector consists out of one or several single inputs. The aim is to build up a model, which predicts the unknown output value given the input vector. A tuple $P = (\mathbf{x}, y)$ with the input vector and the output value is also called *data tuple*.

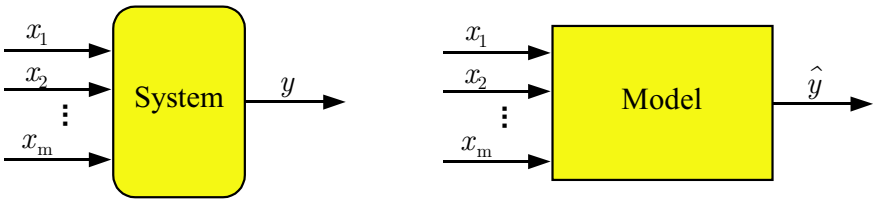


Figure 1.1: The basic task of data-based modelling.

It is assumed, that the underlying input/output behavior of the given system can be described by the so called *true function* $f_{tf}(\mathbf{x})$ as

$$y = f_{tf}(\mathbf{x}) + e \ , \tag{1.1}$$

whereas e is an additional noise term, that cannot be defined exactly. The distribution of the input vectors is given by $\mu_{tf}(\mathbf{x})$. In general both the function f_{tf} and the distribution $\mu_{tf}(\mathbf{x})$ are unknown. The modelling task would be completely solved with knowledge of the function f_{tf} – thus the aim is to learn a good approximation of this function based upon some measured data tuples.

Nominal, Ordinal and Continuous Variables With respect to the possible values, there are three different types of variables.

- *Nominal variables.* Nominal variables can only have symbolic values, that cannot be ordered with respect to a greater-less relation. An example for a nominal variable is the color of an object, which can have the different symbols *red*, *green* and *blue*. It is assumed in this work, that the different symbols of a nominal variable are encoded as integer numbers starting from 1.

- *Ordinal variables.* Ordinal variables can only have symbolic values, but, in contrast to nominal variables, they can be ordered with respect to a greater-less relation. An example for an ordinal variable is a temperature that is measured with the qualitative terms *cold*, *normal*, *warm* and *hot*. Another example is the age of a person that is measured in years. In the context of this work, ordinal variables with just a few different symbols, as the above example with the temperature, are treated as nominal. But ordinal variables with many different symbols, as the above example with the age, are treated as continuous.
- *Continuous variables.* Continuous variables can have arbitrary real values – only limited by the precision of the measuring device. An example for a continuous variable is a temperature measured in centigrade.

Typical Procedure of How to Build up a Data-Based Model The typical procedure of how to build up a data-based model is illustrated in figure 1.2. In practise it is often necessary to execute one or several steps repeatedly to try different approaches and learning algorithms until a suitable solution is found.

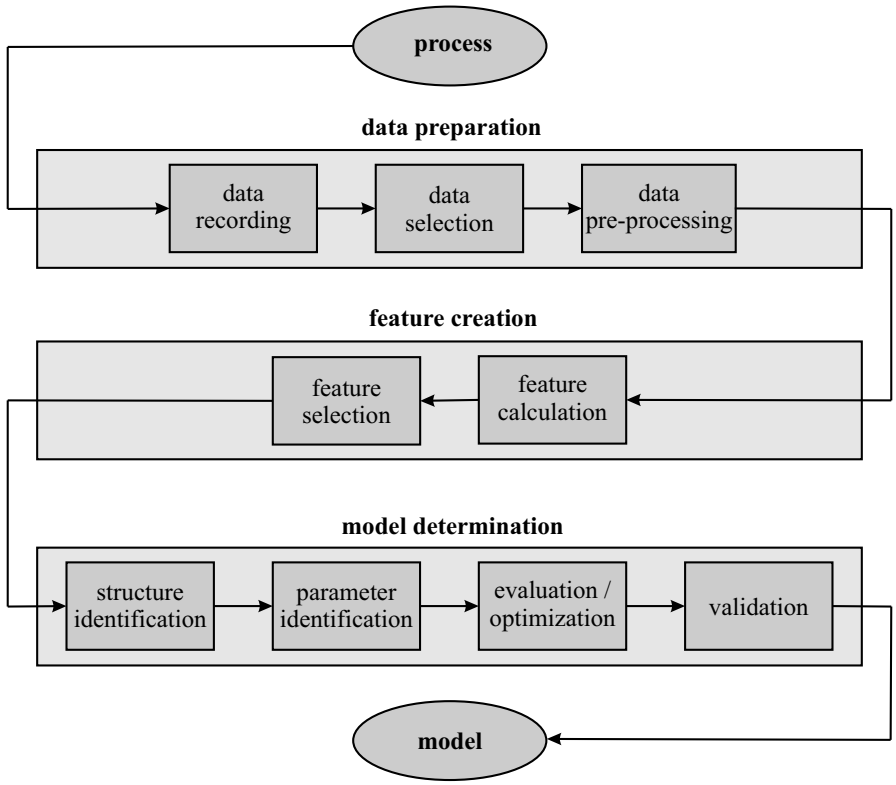


Figure 1.2: Typical procedure of how to build up a data-based model. *Origin:* [Sla01]

- *Data preparation.* First of all, some data is collected from the real system by recording input and corresponding output values for some representative operating conditions. Then outliers or incorrect measurements are removed and some simple pre-processing like standardization or smoothing is done.
- *Feature creation.* The feature creation step is optional. Maybe the measured inputs contain only raw data. Then some features, that are supposed to be more meaningful, are calculated. Afterwards the number of features/inputs may be reduced by using some feature selection strategies. See section 1.6 for a short overview of some data-based approaches.
- *Model determination.* The measured data tuples form the *learn data sample*. The terms *structure* and *model identification* are used in the broader sense here and refer to the selection of a suitable learning algorithm and its application to learn a model from the learn data sample. Where appropriate, some postprocessing is done to fine tune the learned model. Then the model is validated by estimating its prediction accuracy for some new unknown test data tuples. Some notes how to measure a model's prediction accuracy can be found in section 3.1.

Regression versus Classification By the means of the output variable's type there are two fundamentally different types of learning. If the output is nominal, one has got to deal with a classification task, whereas if the output is continuous, one has got to solve a regression problem. This work primarily focuses on classification tasks and solves regression problems using the following trivial approach: The continuous output is transformed, as described in section 1.8.2, into an ordinal output. Then the ordinal output is treated as nominal and thus one gets a classification problem which can be handled as ordinary – of course with some slight modifications of the prediction mechanism.

Batch versus Incremental Learning Learning algorithms can work in a batch or in an incremental way. In the first case, all data tuples are processed simultaneously, whereas in the latter case it is only possible – mostly due to limited computer performance and/or memory – to process the learn data tuples incrementally one after the other without storing the whole training data sample. Most learning algorithms can be realized to work in a batch as well as in an incremental mode. Tendentious, the batch mode yields better results as more information is available at the same time – however at the price of increased computational costs.

Similarity Hypothesis and the Components of a Learning Algorithm The underlying assumptions of most data-based learning algorithms is the so called *similarity hypothesis*, which states, that continuous regions in the input space correspond to the same output class or to similar output values. Thus it is possible to use the knowledge about the output at a particular input position to derive information about probable output values at surrounding input positions.

A data-based learning algorithms consists of the three basic components representation, search and prediction mechanism [Dom97]. These are described in the following itemization.

- *Representation.* This component refers to the way, how the learned knowledge about the system's input/output behavior is operationally stored. Different possibilities to represent this knowledge are, for example, If-Then rules, discriminating hyperplanes or linear additive polynomials.
- *Search.* The search designates the kind, how the model, that best fits a given learn data sample, is chosen from all possible models, i.e. from all models that are possible with respect to the representation used.
- *Prediction Mechanism.* The prediction mechanism predicts the output value given an input position based upon the learned knowledge. Sometimes this component is considered to be a part of the representation.

Due to representation, search and prediction mechanism used, each learning algorithm has a so called *bias*, i.e. the algorithm prefers some generalizations about others and thus will yield better results if the algorithm's bias matches the characteristics of the learning task.

1.2 Rule-Based Models

The fuzzy-logic [Zad65, Zad68] was invented by Zadeh in 1965. This section provides a simple introduction to fuzzy-logic and the functionality of rule-based Mamdani fuzzy models [MG81]. The description is similar to [Kie97]. For a comprehensible online reference see [KHJ97].

Representation Rule-based Mamdani fuzzy models consist of a set of linguistic rules of the form

$$\text{If } \textit{premise} \text{ Then } \textit{consequent} \ . \tag{1.2}$$

The premise contains linguistic statements about single inputs that are combined by a logical And operator and the consequent contains a statement about the output value. Each rule describes qualitatively the system's input/output behavior in a particular part of the input space. In contrast to ordinary hard rules, where a logical expression can only evaluate to 0 and 1, fuzzy-rules can deal with logical expressions, that can evaluate to the continuum $[0, 1]$. A value of 0 resp. 1 denotes that a logical expression is completely fulfilled resp. completely not fulfilled, whereas a value between 0 and 1 denotes that the logical expression is partly fulfilled. The combination of two or more linguistic expressions is done by generalized logical operators. Possible And operators are

$$\begin{array}{lcll} \mu_1 \wedge \mu_2 & = & \min(\mu_1, \mu_2) & \text{(minimum)} \\ \mu_1 \wedge \mu_2 & = & \mu_1 \mu_2 & \text{(algebraic product)} \end{array} \tag{1.3}$$

and possible OR operators[1] are

$$\begin{array}{lcll} \mu_1 \vee \mu_2 & = & \max(\mu_1, \mu_2) & \text{(maximum)} \\ \mu_1 \vee \mu_2 & = & \mu_1 + \mu_2 - \mu_1\mu_2 & \text{(algebraic sum)} \\ \mu_1 \vee \mu_2 & = & \mu_1 + \mu_2 & \text{(ordinarysum).} \end{array} \tag{1.4}$$

Continuous[2] variables are transformed into the degree of membership to a linguistic variable using a so called membership function (MSF). This process is called *fuzzification*. Typical membership functions for the inputs have the form of a triangle, trapezoid, rectangle or gaussian. The output is often encoded as a so called *singleton*, that yields a membership degree of 1 only for a single value and that is 0 otherwise. Figure 1.3 is an example for some membership functions that transform a continuous temperature value into the degree of membership to the linguistic variables *cold*, *normal*, *warm* and *hot*.

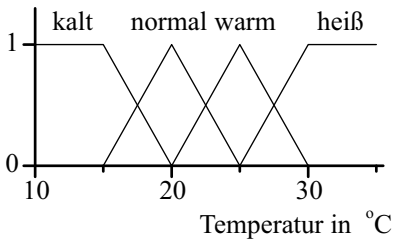


Figure 1.3: Membership functions that transform a continuous temperature value into the degree of membership to a linguistic variable. (Translation: *kalt=cold*, *heiß=hot*) *Origin:* [Sla01]

Prediction Mechanism This paragraph explains, how the rule set is used to predict the output value given an input position. All input values a fuzzified and the activation of each rule is calculated by evaluating the truth value of the premise and combining it with the consequent using an AND operator (inference). Afterwards the activations of all rules are superposed using an OR operator to get a single fuzzy subset for the output (composition). Formally one gets this single fuzzy subset for a set of K rules as

$$\mu(\mathbf{x}, y) = \bigvee_{t=1}^{K} \left(\mu_t(\mathbf{x}) \wedge \mu_t(y)\right) \ , \tag{1.5}$$

whereas $\mu_t(\mathbf{x})$ denotes the truth value of the premise and $\mu_t(y)$ the consequent of the t-th rule. The predicted output value is inferred from $\mu(\mathbf{x}, y)$ by the so called *defuzzification*. The two most commonly used defuzzification methods are the *centroid* method (COG)

$$\widehat{y} = \frac{\int_{-\infty}^{+\infty} y \ \mu(\mathbf{x}, y) \ dy}{\int_{-\infty}^{+\infty} \mu(\mathbf{x}, y) \ dy} \tag{1.6}$$

and the *maximum* method (MOM)

$$\widehat{y} \quad \text{with} \quad \mu(\mathbf{x}, \widehat{y}) = \max \mu(\mathbf{x}, y) \ . \tag{1.7}$$

Metaphorically speaking, the centroid method leads to a kind of compromise of the different values, which are suggested by the rule set, whereas the maximum method is just choosing the most suggested value.

[1] Strictly speaking the ordinary sum is not a fuzzy-logic operator. However it leads to simpler and smoother input/output behavior and thus it is used very often.

[2] The possible symbols of nominal variables are mapped directly to a linguistic variable that can only evaluate to 0 or 1.

How We Chose the Fuzzy Operators and Defuzzification We use the algebraic product as AND operator and the ordinary sum as OR operator. Singletons are used as output membership functions. The centroid defuzzification is used for regression problems, whereas the maximum method is used for classification tasks.

For these settings the defuzzification simplifies to

$$\widehat{y} = \frac{\sum_{t=1}^{K} \mu_t(\mathbf{x})\ y_t}{\sum_{t=1}^{K} \mu_t(\mathbf{x})} \tag{1.8}$$

for the centroid method and to

$$\widehat{y} = y_t \quad \text{mit} \quad t = \arg\max_{t=1}^{K} \mu_t(\mathbf{x}) \tag{1.9}$$

for the maximum method. y_t denotes the particular output value for which the singleton, which is used as output membership function of the t-th rule, has the value 1.

TSK Models TAKAGI, SUGENO and KANG have developed the so called TAKAGI-SUGENO-KANG model (TSK) [TS85]. The significant difference compared with the MAMDANI fuzzy model is, that functions of the input variables instead of fuzzy membership functions are used as consequents. Thus each fuzzy rule defines a local model for a particular part of the input space.

$$\text{IF } \textit{premise} \text{ THEN } y = f(\mathbf{x}) \tag{1.10}$$

The predicted output value $\widehat{y}$ is calculated as a weighted average of the different suggested values analogously to eqn. (1.8). However the value $f_t(\mathbf{x})$ is used instead of a fixed value y_t.

Rating of rules It is possible to assign a rating ϱ to each rule. This rating denotes, how reliable or good is the rule with respect to the learn data sample. The rating is combined directly with the truth value of the premise using the chosen AND operator.

The most commonly used approach is the so called *hitrate*, that is calculated based upon the conditional probability $P(c|p)$ of the consequent c if the premise p is fulfilled as

$$\varrho = P(c|p)\ . \tag{1.11}$$

The simplest way to estimate this probability in the case of a fuzzy rule is

$$\widehat{P}(c|p) = \frac{\sum_{i=1}^{N} \mu_t(\mathbf{x}_i)\ \mu_t(y_i)}{\sum_{i=1}^{N} \mu_t(\mathbf{x}_i)}\ , \tag{1.12}$$

whereas $\mu_t(\mathbf{x}_i)$ and $\mu_t(y_i)$ are the truth values of premise and consequent for the different data tuples i of the learn data sample.

This estimate may be overly optimistic if a premise is only fulfilled for very few learn data tuples. In this case one would expect, that the true hitrate, i.e. the hitrate if an infinite amount of data tuples could be evaluated, is smaller. The often used *laplace correction* [Goo65, Nib87] modifies eqn. (1.12) with respect to the number of possible output symbols S_y to

$$\widehat{P}(c|p) = \frac{1 + \sum_{i=1}^{N} \mu_t(\mathbf{x}_i)\ \mu_t(y_i)}{S_y + \sum_{i=1}^{N} \mu_t(\mathbf{x}_i)}\ . \tag{1.13}$$

If the premise is fulfilled for many learn data tuples, this estimate converges to the uncorrected one.

1.3 Instance-Based Learning

The so called instance-based learning algorithms[3] (IBL) store all or subset of the learn data tuples in memory and use these stored tuples to predict the output for a given input position. Typical for all instance-based learning algorithms is, that – apart from some pre-processing steps – the analysis of the learn data sample is done just at the moment when a prediction for a particular input position is needed. The most commonly known instance-based learning algorithm is the k nearest neighbor (kNN) algorithm [CH67, DH73, Das91]. This algorithm makes a prediction by evaluating the output values of the nearest learn data tuples – the so called nearest neighbors. These nearest neighbors are determined using a suitable distance measure. In the case of $k = 1$ the algorithm simplifies to the simple *nearest neighbor* (NN) algorithm. The evaluation of the output values is done by a kind of voting or by the means of a weighted average. Most often the former is used for classification tasks. Ties must be broken using a suitable tie breaking policy. For example by preferring the output class with the highest a priori probability on the learn data sample[4]. On the other hand the evaluation by the means of a weighted average is often used for regression tasks. A well known approach is the so called *Nadaraya-Watson estimator* [Nad64, Wat64], that can be written for the case $k = N$ as

$$\widehat{y} = \frac{\sum_{i=1}^{N} F(d_i)\ y_i}{\sum_{i=1}^{N} F(d_i)} \ . \tag{1.14}$$

F denotes an arbitrary one dimensional kernel, which is usually chosen to be continuous, bounded and integrable and d_i denotes the distance of the i-th data tuple of the learn data sample to the given input position. Note the similarity of eqn. (1.14) to eqn. (1.8).

The properties of the kNN algorithm resp. the IBL algorithms are [Dom97, WM00a]:

- Advantages
  - easily to understand and simple to implement
  - fast learning, as in the simplest case one only needs to store all learn data tuples
  - complex decision boundaries in the input space can be induced by relatively few learn data tuples
  - very good generalization capabilities for many real problems
- Disadvantages
  - typically high memory consumption
  - relatively slow prediction mechanism, as the distance of a given input position to all stored data tuples needs to be evaluated
  - the resulting model can hardly be interpreted by a human
  - problems may arise if inputs are irrelevant or noisy

Many researchers have developed extension to the kNN resp. to the IBL algorithms. Important and still relevant topics cover instance pruning techniques to reduce the number of stored learn data tuples within a pre-processing step [Aha90, WM97b, WM00b], the development of improved distance measures [SW92, WM96, WM97a] or of – maybe locally adaptively calculated – weighting factors for the input features [WAM97, Aha98]. An other topic, that can be discussed in a broader context of learning algorithms in general, is the usage of feature selection methods. More explanations about distance measures and feature weights can be found in section 1.8. The basics of feature selection methods are introduced in section 1.6.

An approach that, in an extended sense, can be classified as IBL algorithm, is the *exemplar-based generalization* [Sal91, WD95, Dom97]. This approach generalizes the learn data tuples by a kind of merging or generalization to so called *exemplars* or *prototypes*. However this approach leads, in contradistinction to the typically fast IBL

[3] Analogously to [Aha95b, Dom97] this term is used synonymously for the terms *memory-based learning* and *nearest-neighbor learning* and in the extended sense also for the terms *exemplar-based generalization*, *case-based learning* and *kernel-based learning*.

[4] In this work this policy is called *most frequent class* (MFC).

algorithms, to more complex and computationally expensive learning algorithms. The PNC 2 cluster algorithm is based on the exemplar-based generalization approach. The same holds for the NGE and the RISE algorithm, which are described in section 2.4.

1.4 Other Learning Approaches

Two other data-based learning approaches are artificial neural networks and decision trees.

Artificial neural networks [Sar02] are inspired by observations about biological systems. Important approaches are *multi layer perceptron* (MLP) and *radial basis functions* (RBF) networks. In principle neural networks can approximate every computable function. However the learned model is quite complex and can hardly be interpreted by a human. Thus the neuro-fuzzy approach [NK97] combines the learning capabilities of a neural network with the interpretability of a fuzzy-rule set. Therefore a special network structure is used, which can be converted to a fuzzy rule set anytime, i.e. before, during and after the learning.

Decision trees [Qui86] are graphs consisting of nodes and leafs. Each node has one or more following nodes or leafs. A leaf finally contains the classification result. Starting from a so called *root* node, the graph is traversed up to a leaf. A condition is evaluated at each node to decide about the successor that is chosen. For example, a condition for a node with two successors could be the evaluation if a particular input exceeds a threshold. The left successor is chosen if the threshold is exceeded, the right successor otherwise. Decision trees can be transformed into a rule set of – eventually cascaded – IF-THEN-ELSE rules. They are especially good if there are only a few highly relevant inputs.

1.5 Curse of Dimensionality

The term *curse of dimensionality* (COD) was coined by BELLMAN in 1961 [Bel61]. BELLMAN considered the problem to estimate a probability density in a high dimensional feature space based upon a drawn sample. This problem can be seen as the task to estimate the probability density at each cell of a multi-dimensional grid. For a fixed number of grid lines per feature, the number of cells increases exponentially with increasing number of features. Thus the sample size, that is needed to estimate the probability with a pre-specified precision, also increases exponentially.

Nowadays the term COD is used for all kinds of problems that may arise for an algorithm if the number of (input) variables increases. In the context of data-based learning algorithms one has to investigate for each algorithm, how the COD affects runtime, memory requirements and resulting prediction accuracy or required sample size. For example IBL algorithms or similar approaches can suffer from the effect, that, for a given input position, the distance of the nearest neighbor converges towards the distance of the farthest neighbor [HAK00]. Thus it is increasingly difficult to infer anything useful from the nearest learn data tuples.

1.6 Feature Selection

Sometimes only a subset of the available inputs is necessary to learn a model with sufficient prediction accuracy. A preceding selection of the most relevant inputs is often favorably as

- the cost to measure and store the data is reduced
- the resulting model is easier to understand
- the computationally cost to learn the model is reduced, as the runtime of many learning algorithms increases with an increasing number of inputs
- many learning algorithms are susceptible to irrelevant or redundant inputs

Existing feature selection methods can be categorized with respect to the *optimization objective* and *search strategy* used. The optimization objective is a suitably chosen function, that assigns a rating to each particular set of inputs. The search strategy determines, how an input set, that has an optimal or at least a sub optimal rating, is chosen from all possible input sets.

Optimization Objective There are two different approaches with respect to the optimization objective used [JKP94, KJ97, KW00]. On the one hand, there is the so called *filter* approach. For this approach in general one chooses an easy and fast calculable objective – for example the transinformation of the input set to the output. For a short description of how to calculate the transinformation for a single input see section 1.8.2. A formal definition for the calculation of the transinformation for more than one input can be found in [KW00]. One the other hand, there is the so called *wrapper* approach, that uses the prediction accuracy – estimated using cross-validation or a similar approach within the learn data sample – with respect to the particular input set of the learning algorithm itself as optimization objective. The advantage of this approach is, that the bias of optimization objective and learning algorithm are identical. For a filter approach, on the contrary, one cannot guarantee, that an input set with a high rating will also be the most suitable input set with respect to the learning algorithm. However a wrapper approach is computationally more expensive.

Search Strategy Different approaches have been presented how to chose a good input set. The so called *complete* search evaluates all possible input sets and will always find the optimal solution. However, due to the exponentially increasing computational costs, this approach is only feasible for a few inputs. Thus often a so called *greedy* or a so called *genetic* search is used [IV94, DM98]. Known greedy algorithms are forward selection and backward elimination. The former starts with an empty input set and iteratively adds always the input, that best improves the optimization objective. The latter starts with the full set of all inputs and iteratively always removes the input, whose removal has the most positive effect on the optimization objective.

Complete, greedy and genetic search are compared with each other for a wrapper approach with an IBL algorithm and a decision tree in [IV94, DM98]. Often the greedy search already finds a suitable solution, that is only slightly worse compared with the optimal solution. However, for very complex problems, the computationally more expensive genetic search can find a more robust solution.

1.7 Cluster Algorithms

Cluster algorithms [And73, Eve93, JMF00] divide a given set of objects, based upon an suitable similarity measure, in several groups, that are called *clusters*. Thereby the objects within a group should be as similar as possible and the objects from different groups should be as dissimilar as possible. Hard clustering techniques assign each object exclusively to a particular cluster, whereas fuzzy clustering techniques allow, that an objects belongs with a certain membership degree to several clusters at the same time. The objects are mostly given as data tuples, that consist of a m dimensional vector $\mathbf{x}$.

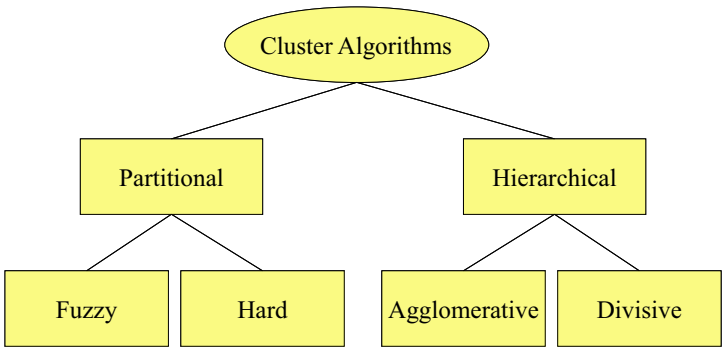


Figure 1.4: Taxonomy of cluster algorithms.

Most cluster algorithms work *unsupervised*, i.e. either they do not use an output at all, or they just treat an existing output as an additional input. Some authors even define cluster algorithms to be unsupervised and otherwise consider an approach as machine learning algorithm[5].

Cluster algorithms can be divided into partitional and hierarchical approaches. The latter produces a complete series of nested partitions, whereas the former only produces a single one. A possible taxonomy of cluster algorithms is shown in figure 1.4. The following paragraph tries to give a short impression about possible similarity measures. Then the partitional and the hierarchical approaches are described in detail.

[5]We do not follow this strict categorization in this work and simply consider the PNC 2 as agglomerative cluster algorithm because this simplifies the explanations.

Similarity Measures A common similarity measure is the use of a distance function[6] like the Euclidean distance for example. Some distance functions like the more general Minkowski metric can be found in section 1.8. Other approaches, for example, evaluate a context of neighboring learn data tuples. Therefor, of course, a distance function is necessary to determine these neighboring tuples. The JARVIS & PATRICK clustering algorithm [JP73] evaluates, how many nearest neighbors are shared by two data tuples. Similar is the so called *mutual neighbor distance* (MND), that is defined as

$$d(\mathbf{x}_a, \mathbf{x}_b) = NN(\mathbf{x}_a, \mathbf{x}_b) + (\mathbf{x}_b, \mathbf{x}_a) \ . \tag{1.15}$$

The function call $NN(\mathbf{x}_a, \mathbf{x}_b)$ resp. $(\mathbf{x}_b, \mathbf{x}_a)$ calculates the rank of the distance of the data vector $\mathbf{x}_a$ resp. $\mathbf{x}_b$ to the data vector $\mathbf{x}_b$ resp. $\mathbf{x}_a$ with respect to the whole data sample. Note, that the MND is not a metric as the triangle inequality is not satisfied.

1.7.1 Partitional Algorithms

Partitional algorithms optimize – usually locally – the partition with respect to an objective function. A commonly known approach is the k-Means algorithm [McQ67], which assigns each data tuple to exactly one of k generated clusters. Thereby k is a pre-specified parameter. The final partition is obtained using a kind of alternating optimization. First of all, the k clusters are initialized somehow. Then repeatedly the assignment of the data tuples is calculated with respect to the actual clusters and then the clusters are updated with respect to the new assignment. The algorithm terminates if a suitable abortion criterion is met – for example if the changes done to the clusters are below a pre-specified threshold. The assignment of the data tuples to the k clusters can be summarized using a so called *partition matrix* $\mathbf{U}$. The element $u_{i,t}$ of this matrix contains the information about the membership of the i-th data tuple to the k-th cluster. Thus the partition matrix has as many rows as there are data tuples and k columns. As the k-Means algorithm makes a hard assignment, the membership degrees can be only 0 and 1. Each cluster is represented by a m dimensional *center* $\mathbf{v}$, whose single components are determined given the partition matrix and the data tuples as

$$v_{t\,j} = \frac{\sum_{i=1}^{N} u_{i,t}\ x_{i\,j}}{\sum_{i=1}^{N} u_{i,t}} \ . \tag{1.16}$$

The partition matrix in turn is calculated by assigning each data tuple to the cluster, whose center is nearest to the data tuple. The prototypical algorithm of partitional clustering is summarized in table 1.1.

Table 1.1: Prototypical algorithm of partitional clustering.

1.	Randomly initialize the k clusters
2.	Recalculate the partition matrix with respect to the actual clusters
3.	Recalculate the clusters with respect to the actual partition matrix
4.	Proceed with the 2nd step until an abortion criterion is met

It can be shown, that the k-Means algorithm locally minimizes the objective function

$$J = \sum_{t=1}^{K} \sum_{i=1}^{N} u_{i,t}\ d_{i,t}^2 \qquad \text{with} \qquad d_{i,t} = ||\mathbf{x}_i - \mathbf{v}_t|| \ , \tag{1.17}$$

whereas $d_{i,t}$ denotes the suitably measured distance of the data vector $\mathbf{x}$ to the center of the t-th cluster.

[6] Strictly speaking, a distance function is a dissimilarity measure, that must be inverted to serve as a similarity measure.

Fuzzy Clustering The Fuzzy-C-Means (FCM) and the Gustafson Kessel (GK) algorithm [Bez81][GK79] extend the k-Means algorithm for the case of fuzzy clustering by extending the possible membership values of a data tuple to a cluster from the boolean values 0 and 1 to the interval $[0, 1]$. We now focus on the probabilistic approach, that limits the sum of membership degrees of each data tuple as

$$\sum_{t=1}^{K} u_{i,t} = 1 \ . \tag{1.18}$$

Now the clusters are calculated for a given partition matrix as

$$v_{t,j} = \frac{\sum_{i=1}^{N} u_{i,t}^{\alpha} x_{i,j}}{\sum_{i=1}^{N} u_{i,t}^{\alpha}} \ , \tag{1.19}$$

whereas α is a so called fuzziness parameter that is typically chosen as 1 or 2. The partition matrix given the clusters is calculated as

$$u_{i,t} = \frac{1}{\sum_{w=1}^{K} \left(\frac{d_{i,t}}{d_{i,w}} \right)^{\frac{2}{\alpha - 1}}} \ , \tag{1.20}$$

whereas $d_{i,w}$ is the distance of the i-th data tuple to the w-th cluster. For the FCM algorithm this distance is typically calculated as the Euclidean distance of the data tuple to the cluster's center, whereas the GG algorithm uses a kind of local Mahalanobis distance as

$$d_{i,w}^2 = \sqrt[m]{\det(\mathbf{Q}_w)} \ (\mathbf{x}_i - \mathbf{v})^T \ \mathbf{Q}_w^{-1} \ (\mathbf{x}_i - \mathbf{v}_w) \tag{1.21}$$

with $\mathbf{Q}$ being the covariance matrix[7]

$$\mathbf{Q}_t = \sum_{i=1}^{N} u_{i,w}^{\alpha} (\mathbf{x}_i - \mathbf{v}_w)^T (\mathbf{x}_i - \mathbf{v}_w) \tag{1.22}$$

of the data tuples assigned to the w-th cluster.

The FCM algorithm is only capable of representing spherical clusters, whereas the GG algorithm, due to the additional covariance matrix that characterizes each cluster, can represent ellipsoidal clusters of the same size. For the sake of completeness we mention the Gath and Geva (GG) algorithm [GG89], that characterizes each cluster by an additional size parameter and is thus capable of representing ellipsoidal clusters of different sizes. However, if the complexity of the cluster representation increases, the algorithms tends to stuck in a local optimum and thus a good clustering result depends more and more on a good initialization [HKK97].

1.7.2 Hierarchical Algorithms

Hierarchical cluster algorithms can be further subdivided into the so called *agglomerative* and *divisive* approaches. An agglomerative approach starts with each data tuple belonging to a single cluster and then iteratively merges the two clusters, that are closest to each other with respect to a suitably chosen similarity measure. The process is terminated, if all data tuples belong to a single cluster or if an abortion criterion is met. On the other hand, divisive algorithms start with a partition, where all data tuples belong to a single cluster and then iteratively divide clusters, until each data tuple belongs to its own cluster or until an abortion criterion is met. Most hierarchical cluster algorithms are agglomerative – this approach is simpler as the divisive approach additionally needs a strategy of how to divide a cluster, i.e. how to apportion the data tuples of the actually processed cluster to the two new clusters. The next paragraph describes some agglomerative cluster algorithms. For divisive approaches see [SBBG02].

[7] Normalization terms are omitted as eqn. (1.21) already normalizes to the determinant of the covariance matrix.

The probably most commonly known agglomerative approaches are the *single* and the *complete linkage* algorithm [SS73][Kin67]. These algorithms iteratively chose two clusters to merge as follows: First of all, the minimal resp. the maximal distance of each possible pair of two arbitrary data tuples from the clusters C_a and C_b is calculated as

$$d(C_a, C_b) = \min_{w \in C_a \; z \in C_b} d(\mathbf{x}_w, \mathbf{x}_z) \qquad \text{resp.} \qquad d(C_a, C_b) = \max_{w \in C_a \; z \in C_b} d(\mathbf{x}_w, \mathbf{x}_z) \; . \tag{1.23}$$

Thereby $w \in C_a$ resp. $z \in C_b$ denotes the set of all data tuples of the cluster C_a resp. C_b. Then the pair C_a and C_b is chosen, for which the distance $d(C_a, C_b)$ is minimal. The single linkage algorithm tends to produce clusters, that are rather elongated chains [Nag68], whereas the complete linkage algorithm prefers compact clusters [FBY92].

Other variants are the *average distance* and *Ward's* algorithm [Mur83][War63]. The former algorithm is similar to the single and to the complete linkage algorithm, but, instead of eqn. (1.23), the average distance of all possible pairs of two arbitrary data tuples from the two clusters C_a and C_b

$$d(C_a, C_b) = \frac{1}{|C_a||C_b|} \sum_{w \in C_a} \sum_{z \in C_b} d(\mathbf{x}_w, \mathbf{x}_z) \tag{1.24}$$

is used. Thereby $|C_a|$ resp. $|C_b|$ denotes the number of data tuples assigned to the particular cluster. The latter algorithm chooses in each step the pair, whose merging leads to the smallest increase of the sum of squared distances from the data tuples to the particular cluster, they are assigned to. Thereby the distance of a data tuple to a cluster is defined similar to the k-Means algorithm using a center calculated according to eqn. (1.16) to characterize each cluster. Ward's algorithm prefers, as the k-Means algorithm, spherical clusters.

1.8 Distance Functions

Distance functions are used to measure the distance between two data tuples P_a and P_b. In this work distances are always calculated in the input space. Thus the term *input vector* or just *input* is used in the following descriptions. In the context of unsupervised clustering these terms should be replaced by *data vector* and *variable*. A broadly known distance function is the *Minkowski* metric, that is defined as

$$d = \sqrt[\rho]{\sum_{j=1}^{m} d_j^{\rho}} \; . \tag{1.25}$$

For continuous inputs the component wise distances d_j are calculated as

$$d_j = |x_{aj} - x_{bj}| \; , \tag{1.26}$$

whereas x_{aj} resp. x_{bj} denotes the j-th component of the input vector $\mathbf{x}_a$ resp. $\mathbf{x}_b$. For some particular values of the metric parameter ρ, the Minkowski metric corresponds to the special distance functions as stated in table 1.2.

Table 1.2: Parameter values ρ for the Minkowski metric.

Block wise distance	$\rho = 1$	$d = \sum_{j=1}^{m} d_j$
Euclidean distance	$\rho = 2$	$d = \sqrt{\sum_{j=1}^{m} d_j^2}$
Chebychev distance	$\rho = \infty$	$d = \max_{j=1}^{m} \lvert d_j \rvert$

Usually the metric parameter is chosen to satisfy the inequality $\rho \geq 1$. However, also other values are possible. In [AHK01] the so called *fractional distance metric*, that allows values $0 < \rho < 1$, is introduced and it is shown experimentally, that this metric can improve the prediction accuracy, if applied in a high dimensional input space together with a kNN algorithm.

Component Wise Distances for Nominal Inputs The simplest approach to calculate the component wise distance d_j of the two symbols $w = x_{aj}$ and $z = x_{bj}$ of a nominal input x_j is the so called *overlap* metric. The distance is 0 if the to symbols coincide and 1 otherwise.

$$d_j = \begin{cases} 0 & x_{aj} = x_{bj} \\ 1 & else \end{cases} \tag{1.27}$$

In [SW92] the so called *Value Difference Metric* (VDM) is introduced to allow a more precise graduation of the calculated distances. The idea of the VDM metric is, that two symbols $w = x_{aj}$ and $z = x_{bj}$ of a nominal input x_j are closer to each other, if the conditional probabilities $P(y = s|x_j = w)$ and $P(y = s|x_j = z)$ are similar for the different possible output classes s. A simplified VDM metric can be calculated as

$$d_j = \sum_{s=1}^{S_y} |P(y = s|x_j = w) - P(y = s|x_j = z)| = \sum_{s=1}^{S_y} \left| \frac{N_{w,s}}{N_w} - \frac{N_{z,s}}{N_z} \right| . \tag{1.28}$$

The required probabilities are estimated based upon the frequentness with respect to the given learn data sample. N_w resp. N_z is the number of learn data tuples, for which the input x_j has the symbol w resp. z and $N_{w,s}$ resp. $N_{z,s}$ is correspondingly the number of learn data tuples, for which additionally the output has the symbol s. Remind, that the different possible output symbols are encoded as integer values starting from 1.

If there are several nominal or mixed nominal and continuous inputs, the component wise distances d_j are calculated according to eqn. (1.27) or (1.28) for nominal and according to eqn. (1.26) for continuous inputs. Afterwards the component wise distances are summarized to the overall distance using eqn. (1.25). In the case of $\rho = 2$ the resulting distance measures are called *Heterogeneous Euclidean Overlap Metric* (HEOM) and *Heterogeneous Value Difference Metric* (HVDM) [WM97a][8].

Missing Feature Values Some practical learning tasks contain input vectors with missing feature values. There are several approaches how to deal with this problem. The rule induction algorithm Rise [Dom97], for example, treats a missing feature value as an additional symbol. The representation of the rules is extended by an additional flag for each input, that indicates, if missing feature values are covered by the rule's premise. Note, that this approach is possible independently of the input's type. Another approach is the ignore strategy [Aha90], that simply ignores a component wise distance in eqn. (1.25) by setting $d_j = 0$, if a feature value is missing. The overall distance is afterwards divided by the number of known features to distinguish a perfect match from the case of an input vector, where all feature values are missing. Alternatively the data sample can be prepared by guessing missing values – for example by using a special model, that was learned for this purpose. An overview of several approaches used together with decision trees can be found in [Qui89, LWTB97].

1.8.1 Normalization

Continuous Inputs All inputs should contribute evenly to the overall distance value [9]. However this requirement is not always satisfied for the distance function (1.25). For continuous inputs a problem arises, if the ranges of the different inputs differ significantly. For example, if one input is within a range of [0..1] and the other within the range of [0..100], then the latter input can dominate the other one. Thus it is advisable to normalize the single inputs by a factor δ_j, so that the component wise distances are mostly within the interval $[0, 1]$. The normalization can be integrated into the distance function.

$$d = \sqrt[\rho]{\sum_{j=1}^{m} \left(\frac{d_j}{\delta_j} \right)^{\rho}} \tag{1.29}$$

[8] Recently published work [WM97a, WM00a] reports about the successful application of the VDM and the similar and further developed *Interpolated Value Difference Metric* (IVDM) in the context of data-based learning algorithms. However first experiments with the Pnc 2 cluster algorithm were disappointing, as the prediction accuracy – compared with a properly normalized and weighted heterogenous Minkowski overlap metric as defined by eqn. (2.15 – could not be improved. Thus we discarded the approach of the VDM or IVDM metric, as it is more complicated.

[9] If some inputs are more relevant than others one can use feature weights as described in section 1.8.2.

An easy approach to chose the normalization factors is to couple them to the range or standard deviation – calculated using the learn data sample – of the particular input. For a formal definition of these simple statistics see table 1.3.

Table 1.3: Simple statistics of an input x.

Mean	$\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i,$
Standard deviation	$\sigma = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2$
Maximum value	$x_{max} = \max_{i=1}^{N} x_i$
Minimum value	$x_{min} = \min_{i=1}^{N} x_i$
Range	$x_{range} = x_{max} - x_{min}$

If the normalization factors are chosen as

$$\delta_j = x_{range\,j}\ , \tag{1.30}$$

then all possible component wise distances d_j of two learn data tuples are guaranteed to be within the interval $[0, 1]$. However the inputs for a test data sample can have a different range and thus it is possible to get a component wise distance $d_j > 1$ – but usually this should not be a problem as a slightly higher component wise distance is often justified in such a case. A problem when using the range normalization is, that this statistic can be highly affected by noise. Thus often a certain percentage of the smallest and biggest values is omitted when calculating the range.

If the normalization factors are chosen as the fourth of the standard deviation

$$\delta_j = 4\sigma_j\ , \tag{1.31}$$

this leads to component wise distances, that are mostly in the interval $[0, 1]$. This is due to the fact, that 95% of all values of a normally distributed variable x_j are inside the interval $[-2\sigma_j, +2\sigma_j]$ and thus the distance of two randomly chosen values will mostly be less than $4\sigma_j$.

Mixed Continuous and Nominal Inputs Proper normalization for mixed continuous and nominal inputs is a non trivial problem, if the overlap metric according to eqn. (1.27) is used to calculate the component wise distances of nominal inputs[10]. The component wise distances are within the interval $[0, 1]$, but the average value tends to be much bigger than the average value of a normalized continuous input. This leads to an disproportionate high influence of nominal inputs. In the context of this work, two different methods how to normalize mixed continuous and nominal inputs are considered.

The first and simplest approach is to keep on using eqn. (1.30) or (1.31) to determine the normalization factors for continuous inputs and to simply chose a fixed $\delta = \delta_{Symb} > 1$ for all nominal inputs.

The second approach is to calculate all normalization factors – the ones for the nominal as well as the ones for the continuous inputs – using the normalization by the *average component wise distance* as it is described in [HC99]. The basic idea of this approach is to normalize each input, such that the average normalized component wise distance is 1. Therefore the average component wise distance of two learn data tuples to each other are calculated as

[10] For notes on how to proper normalize distances, that were calculated using eqn. (1.28) see [WM97a].

$$\delta_j = d_{avr} = \frac{\sum_{i=1}^{N} \sum_{z=i+1}^{N} d_j(x_{i,j}, x_{z,j})}{\frac{1}{2} N(N-1)} , \tag{1.32}$$

whereas N is the sample size and $d_j(x_{i,j}, x_{z,j})$ denotes the distance of the i-th to the z-th learn data tuple. For large sample sizes N it is not necessary to evaluate all possible pairs of two learn data tuples to get a sufficiently robust estimate – the computationally costs can be reduced by only evaluating a subset of the learn data sample.

1.8.2 Feature Weights

The influence of more relevant inputs can be increased using so called *feature weights*. Do not confuse with the different aims of normalization and weighting. The normalization tries to balance the influence of the inputs, whereas the feature weights afterwards scale each input's influence with respect to its relevance.

Several methods to calculate feature weights with application to IBL algorithms are described in [WAM97, Aha98]. The following paragraph describes the approach to calculate feature weights based upon the mutual information [Rez94] of a particular single input to the output. This approach has also been used in [WD95] together with the NGE and the kNN algorithm.

The mutual information $I(x, y)$ measures the decrease of the uncertainty about the outcome of a discrete random variable y, if another discrete variable x is known. The uncertainty is measured by the entropy $E(y)$ [Sha48].

$$E(y) = -\sum_{s=1}^{S_y} P(y = s) \log P(y = s) \tag{1.33}$$

$$I(x, y) = I(y, x) = \sum_{w=1}^{S_x} \sum_{s=1}^{S_y} P(y = s \wedge x = w) \log \frac{P(y = s \wedge x = w)}{P(y = s) P(x = w)} \tag{1.34}$$

Thereby S_x and S_y denotes the number of possible discrete values of the random variable x resp. y and $P(x = w)$ and $P(y = s)$ denote the probability of the w-th resp. the s-th value. A division by zero is circumvented by setting $\log \frac{0}{0} = 0$. If an input does not contribute to reduce the uncertainty about the output, then the mutual information of this input will be 0. If an input completely explains the output, then the mutual information equals the entropy of the output.

The mutual information can be easily calculated and the resulting values are relatively robust with respect to different learn data samples.

The feature weights are calculated based upon the mutual information as follows: The mutual information $I(x_j, y)$ is calculated for each of the m inputs. Continuous inputs are discretized – as described in the following paragraph – for this calculation. The values $I(x_j, y)$ are normalized to get an average feature weight of one.

$$\omega_j = \frac{m\, I(x_j, y)}{\sum_{w=1}^{m} I(x_w, y)} \tag{1.35}$$

Discretization of Continuous Inputs Continuous inputs can be transformed into ordinal inputs using a so called discretization method, that divides the range of the continuous input into several intervals. Each continuous value is replaced by the number of the interval it is in[11]. The simplest and most commonly used approaches are the so called *equidistant* and the *equifrequent* discretization. Both approaches work univariate and unsupervised. It is necessary to pre-specify a fixed number of discretization intervals N_{Bins}.

- *Equidistant discretization.* The range of the input x is equidistantly divided into N_{Bins} intervals of the width $w = \frac{x_{range}}{N_{Bins}}$. As by the range normalization it is advisable, to ignore a small percentage of the smallest and highest feature values when determining the input's range to get a more robust estimate in the case of noise. The left- and the right-most interval are extended to infinity.

[11] We principally enumerate these intervals starting with 1.

- *Equifrequent discretization.* The equidistant discretization can be unfavorable if the numbers of data tuples, that fall into each interval, differ significantly. The equifrequent discretization faces this problem by positioning the intervals such that each interval contains approximately the same number of data tuples $N' = floor(\frac{N}{N_{Bins}})$. Thereby the operator $floor(x)$ returns the largest integer not greater than x, i.e. rounds down. The left- and the right-most interval are extended to infinity.

Several more advanced discretization approaches are published. Of particular interest for application together with the calculation of the mutual information is the entropy based method first published in [FI89]. For an overview about possible other approaches see [HLTM99, DKS95].

The method in [FI89] produces a discretization, that minimizes the entropy of the output y, that is calculated according to eqn. (1.33) for each interval. It is not necessary to pre-specify the number of intervals. The approach is applied as follows: The input is divided into two intervals, such that the entropy of this discretization is minimal. Afterwards, one continues recursively with the resulting intervals as long as the *Minimum Description Length Principle* (MDLP) is satisfied. This principle weights the reduction of the entropy against the increasing complexity of the discretization. The MDLP was originally introduced in the field of communication theory in the context of the bandwidth needed to transmit a message from a sender to a receiver.[12]

1.9 Cluster Algorithms in the Context of Data-Based Modelling

The data-based modelling aims at finding continuous regions in the input space, where the output values of the learn data tuples have the same, or at least similar, output values. Cluster algorithms can be used in this context in various ways. It is possible to treat each cluster as a single rule. Or the cluster algorithm is used as a kind of pre-processing step to provide an advantageous initialization for a learning algorithm or to reduce the learning algorithm's space and/or time consumption.

Direct Rule Generation This paragraph gives a short sketch how a cluster algorithm can be used for direct rule generation. If the cluster algorithm works unsupervised, then it can either be applied to the input or to the input/output space. The output is treated as additional input in the latter case. However we only consider the first case in the following descriptions. For a quick introduction see [KK97].

Each cluster corresponds to a rule of the form

$$\text{If } \textit{input vector is inside the cluster} \text{ Then } y = c\ . \tag{1.36}$$

Partitional cluster algorithms usually use prototypes to represent the clusters. The truth value of the premise can then be determined using the distance of the given input vector to the prototype as the argument of a suitably chosen kernel function.

Hierarchical cluster algorithms in contrast do not always use an explicit cluster representation, as they only need to store the assignments of the data tuples to the clusters. Like described in section 2.2.1, it is possible to use the cuboid, that encloses all data tuples, that are assigned to the cluster, as premise.

The consequent requires the specification of a fixed output value c. For the k-Means or Fcm algorithm, for example, it is possible to calculate this value as a kind of mean of the output values of the data tuples assigned to the cluster. Or, as for the Pnc 2 cluster algorithm, the way the algorithm works predetermines the output value right from the start.

Usage as Pre-Processing Algorithm Basically there are three different approaches how to use a cluster algorithm to pre-process the learn data sample for a subsequently executed learning algorithm.

- *Combination with evolutionary or genetic algorithms.* The cluster algorithm is used to generate a first rule set, which is then optimized using an evolutionary or genetic algorithm. Insofar the cluster algorithm is used to find a good initialization. Concrete examples of this approach can be found in [BGC97, GSJ97, RSA00].

- *Combination with artificial neural networks.* The cluster algorithm is used, in the same manner as for the combination with evolutionary or genetic algorithms, to generate a first rule set. This rule set is then

[12] This approach seems to be especially qualified to be used in the context of mutual information calculations. However, first experiments with the Pnc 2 cluster algorithm were disappointing as the prediction accuracy was not increased. Thus this approach was discarded.

transformed into a neural network. Therefor a special network structure is used, that can be transformed into a rule set and back again at any time [NK97]. The rule set can then be optimized employing neural network learning strategies and capabilities. Concrete examples of this approach can be found in [Fri97, RPP99].

- *Usage to generate TSK models.* The cluster algorithm is used to produce a partition of the learn data sample, or, to be more concrete, to identify regions of the input space, for which local models are learned. An introductory description of this approach can be found in [Bab98, Bab02]. For an example see [HK00].

1.10 Goals of this Work

This work aims to develop a cluster algorithm, that can be used for direct generation of rules of the form (1.2). Therefor unsupervised hierarchical agglomerative cluster strategies shall be extended to meet the different goal of rule generation, namely the identification of continuous regions in the input space, that have the same or at least similar output values. Thereby – apart from a high prediction accuracy of the learned rule sets – the following conditions should be considered.

- *Interpretable rules.* For maximal interpretability of the resulting rule set, each rule should describe a relevant part of the input/output behavior of the system, i.e. it should be locally reasonable. Therefore it is necessary to decide upon each cluster using only local decision criterions. Note the difference to the approaches outlined in section 1.9, where the cluster algorithm is used in combination with evolutionary and genetic algorithms or neural networks, which optimize the rule set using a global objective. In principle it would also be possible to employ such strategies, but this will be only noted in the margin here.
- *High dimensional input spaces.* The developed cluster algorithm should be applicable in high dimensional spaces of 40 and more inputs. Therefore it is necessary to analyze, how the COD problem affects the algorithm. Then suitable mechanism to circumvent the COD problem must be developed.
- *Fully automated parameter tuning.* Many learning algorithms contain some free parameters, that must be specified by the user and allow the adjustment of the algorithm's strategies to the characteristics of the actually given problem. Usually a good adjustment of these parameters is only possible, if the user has a deep understanding of the particular learning algorithm. Thus a fully automated self configuration of all settings is of great value for an unexperienced user.

Chapter 2

The PNC 2 Algorithm

This chapter describes the *Positive and Negative example-based Clustering* (PNC 2) algorithm. First the basic idea is outlined. Then the simplified version of the algorithm is described and the pseudo-code is presented. Afterwards advanced details and enhancements are given and the most similar existing algorithms are identified and compared with the PNC 2 cluster algorithm.

2.1 Outline

The PNC 2 is a supervised hierarchical agglomerative cluster algorithm. Agglomerative cluster algorithms start treating each data tuple as an own cluster and then merge iteratively two clusters with each other until an abortion criterium is met or until all data tuples are merged within one single cluster. In most agglomerative cluster algorithms the clusters to be merged are chosen based upon a distance or point density criterium evaluated in the (input) space. Thus these algorithms are capable to capture the structure of the data in the (input) space, but they do not consider, if the merged data tuples have the same or at least a similar output value. The PNC 2 in contrast executes a test before merging. This test evaluates if the merged cluster is a valid rule.

Each cluster is represented by an axis-parallel *cuboid*[1] given in the input space and by an associated output value. The cuboid is build to include all the input vectors of the data tuples merged in the cluster. A cluster can be transformed to an IF-THEN rule: The cuboid corresponds to the premise and the associated output value is taken as the conclusion.

The agglomerative clustering is done as follows: First of all, each data tuple is considered as an own cluster by taking the data tuple's output as the cluster's output value and by building a *trivial* cuboid that includes only the data tuple's input vector. Only clusters with the same output value can be merged. Thus the clusters are assorted in groups according to their output value and it is tried to merge as many clusters as possible within each group. Therefor iteratively the two clusters closest to each other and not previously tested are chosen and, by the means of a so called *merge test*, it is evaluated if the merged cluster is a valid rule. The clustering within a group is finished if there are no possible pairs of two clusters not tested anymore.

The output for a given input position is predicted by evaluating the output values of the nearest clusters.

2.2 Simplified Algorithm

This section describes the simplified but in low-dimensional input spaces fully operational basic algorithm. Several restrictions are made to be able to present a compact and comprehensible version. So, first of all, only continuous inputs are considered and the aspects of normalization and weighting are neglected. The description is further restricted to classification tasks with a simplified prediction mechanism. The procedure for regression tasks, special mechanisms to cope with the COD problem and additional details and enhancements are described in section 2.3.

[1] The term *cuboid* fits for continuous inputs only. For a detailed description of how to represent nominal inputs see chapter 2.3.1.

2.2.1 Representation

Cluster A data tuple is given, as described in section 1.1, by a m dimensional input vector $\mathbf{x}$ and an associated output value y. A cluster is represented by an output value c and by a cuboid H. This cuboid is described by its lower left and its upper right edge $\mathbf{l}$ resp. $\mathbf{r}$ in the m dimensional input space. A data tuple can be transformed to an elementary cluster by taking the data tuple's output as the cluster's output value and by building a *trivial* cuboid that only includes the data tuple's input vector.

$$\mathbf{l} = \mathbf{x} \qquad \mathbf{r} = \mathbf{x} \qquad c = y \tag{2.1}$$

Distance Measure The distance of two clusters C_a and C_b is the distance of the two cuboids H_a and H_b, which is estimated as the sum of the component wise distances of the cuboids' outer edges. This is equivalent to a Minkowski metric like in eqn. (1.25) with $\rho = 1$. With respect of a single input j, the cuboid is an interval, that is determined by the left and right bound l_j resp. r_j. The component wise distance d_j of two cuboids is 0 if, and only if, the two intervals overlap. Thus the overall distance of two cuboids is 0, if the two cuboids overlap with respect to all inputs.

$$d_j = \begin{cases} l_{aj} - r_{bj} & l_{aj} > r_{bj} \\ l_{bj} - r_{aj} & r_{aj} < l_{bj} \\ 0 & \text{else} \end{cases} \tag{2.2}$$

As the input vector of a data tuple can be transformed into a trivial cuboid, eqn. (1.25) and (2.2) can also be used, to determine the distance of a data tuple to a cluster. Note, that the distances are determined in the input space only and thus the output values of the data tuple and the cluster are irrelevant. A distance of $d = 0$ means, that the data tuple lies inside the cuboid: The data tuple is *covered* by the cluster.

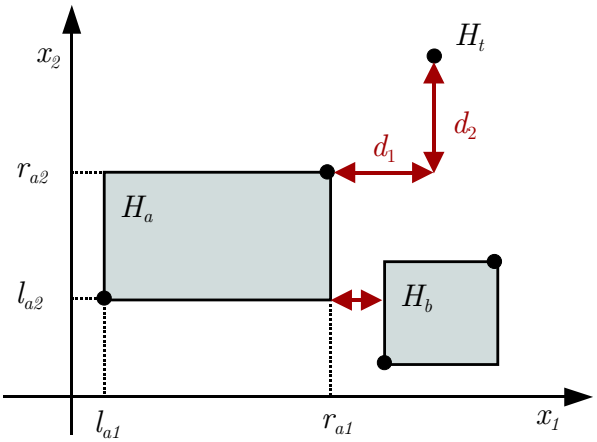


Figure 2.1: Representation and distances in a two dimensional input space.

From Clusters to Rules A cluster $C = (H, c)$ is equivalent to an If-Then rule of the form (2.3). The input space covered by the cuboid is used as premise and the cluster's output value is taken as conclusion.

$$\text{If } \mathbf{x} \in H \text{ Then } y = c \tag{2.3}$$

An input vector $\mathbf{x}$ is inside a cuboid, if its distance to the cuboid gets $d(\mathbf{x}, H) = 0$. Thus the logical value of the premise is determined by the membership function

$$\mu(\mathbf{x} \in H) = \begin{cases} 1 & d(\mathbf{x}, H) = 0 \\ 0 & else \end{cases} . \tag{2.4}$$

By projecting this membership function to the single inputs it is possible to get the more familiar form

$$\text{If } x_1 = \mu_1 \text{ And } x_2 = \mu_2 \text{ And } \ldots \text{ And } x_m = \mu_m \text{ Then } y = c \tag{2.5}$$

with

$$\mu_j = \begin{cases} 1 & d_j = 0 \\ 0 & else \end{cases} . \tag{2.6}$$

Note, that only crisp membership functions are used. Thus an input position is either completely inside or completely outside of a cuboid. Section 2.3.5 describes how to alternatively transform the clusters to fuzzy rules.

Generalization The cuboid of a cluster shall include all input vectors of the data tuples merged within the cluster. Thus two clusters C_a and C_b with the same output value are merged to a generalized cluster C_g by just copying the output value and building a generalized cuboid H_g from the two cuboids H_a and H_b as follows: Take the component wise minima as lower left and the component wise maxima as upper right bound. This is the most specific generalization with respect to the representation used.

$$\mathbf{l}_g = \min(\mathbf{l}_a, \mathbf{l}_b) \qquad \mathbf{r}_g = \max(\mathbf{r}_a, \mathbf{r}_b) \qquad c_g = c_a = c_b \tag{2.7}$$

Let the number of *positive* and *negative* examples of each cluster be defined as follows.

- All data tuples, that are inside a cluster's cuboid and whose output value coincides with the cluster's output value, i.e. all data tuples for that both the premise and the conclusion of the associated rule hold, are counted as positive examples p of that cluster. The number of data tuples merged within the cluster is used as a simple and efficiently determinable approximation – although it may be a little bit too pessimistic. In the following the term *mass* is used synonymously.
- All data tuples, that are inside a cluster's cuboid and whose output value does not match the cluster's output value, i.e. all data tuples for that the premise but not the conclusion of the associated rule hold, are counted as negative examples n of that cluster.

2.2.2 Search

Overview The algorithm is initialized by transforming each data tuple to an elementary cluster. Only clusters with the same output value can be merged with each other. Thus the clusters are assorted in groups with the same output value. Each group is processed one after the other and it is tried to iteratively merge as many clusters as possible. A single cluster step is done as follows: The two clusters closest to each other and not previously tested are chosen, generalized and based upon a so called *merge test* it is checked, if the generalized cluster is a valid rule. If the test is passed, the two old clusters are replaced by the new generalized one. The search is finished if in all groups all possible pairs of the remaining clusters have failed the merge test. To further reduce the number of clusters, all clusters whose mass does not exceed a pre-specified threshold p_{min} are deleted. The parameter p_{min} is also called *minimal mass* and the number of clusters before and after this reduction is called K resp. K_{Red}.

The Doubled Merge Test The generalized cluster should be a valid rule. A rule is rated by its hitrate, that is approximated as the rate of the positive to the overall number of examples covered by the rule as

$$\varrho = \frac{p}{p+n} \qquad (ordinary\ hitrate) \tag{2.8}$$

resp.

$$\varrho = \frac{1+p}{S_y+p+n} \qquad \text{(laplace corrected hitrate)} . \tag{2.9}$$

The maximal number of negative examples n_{max}, that the generalized cluster may have, is determined based upon the number of positive and negative examples of the two clusters a and b as

$$n_{max} = \min(\, p_a\eta + n_b,\ p_b\eta + n_a \,) \qquad 0 \le \eta \le 1 . \tag{2.10}$$

The parameter η controls how many negative examples may be covered relative to the number of positive examples of the clusters a and b. No negative examples are permitted if η is 0. A value $\eta > 0$ may be useful in noisy or contradictory domains.

Following eqn. (2.10) is explained. Look at the two clusters a and b and the cluster g that is generalized out of the two. The clusters' cuboids cover the input space H_a, H_b and H_g. Equivalent to the merged cluster is, corresponding to eqn. (2.3), the rule

$$\text{If } \mathbf{x} \in H_g \text{ Then } y = c_g \tag{2.11}$$

But not this rule but the two rules

$$\text{If } \mathbf{x} \in H_g \text{ And } \mathbf{x} \notin H_a \text{ Then } y = c_g \tag{2.12}$$

and

$$\text{If } \mathbf{x} \in H_g \text{ And } \mathbf{x} \notin H_b \text{ Then } y = c_g \ . \tag{2.13}$$

are tested and rated. The parameter η yields a threshold of the minimum hitrate ϱ_{min}, that both rules must exceed in order to pass the merge test.

$$\varrho_{min} = \frac{1}{1+\eta} \tag{2.14}$$

This procedure prevents an unwanted bloating of cuboids with a bigger mass if they are generalized and tested with cuboids with a smaller mass. Look at the situation illustrated in figure 2.2, where the rule equivalent to the generalized cluster exceeds a minimum hitrate but however a generalization is undesirable. Shown is the 2 dimensional input space with the two cuboids H_a and H_b of the two clusters a and b. Between the two cuboids are several data tuples with a different output value that are all negative examples of the generalized cluster. If the parameter η is chosen to be 1, this yields a minimum hitrate of $\varrho_{min} = 50\%$. The generalized cluster would be a valid rule, but a merging is not desired as the generalized premise makes – from the viewpoint of the bigger cluster – an invalid prediction for the additionally covered input space.

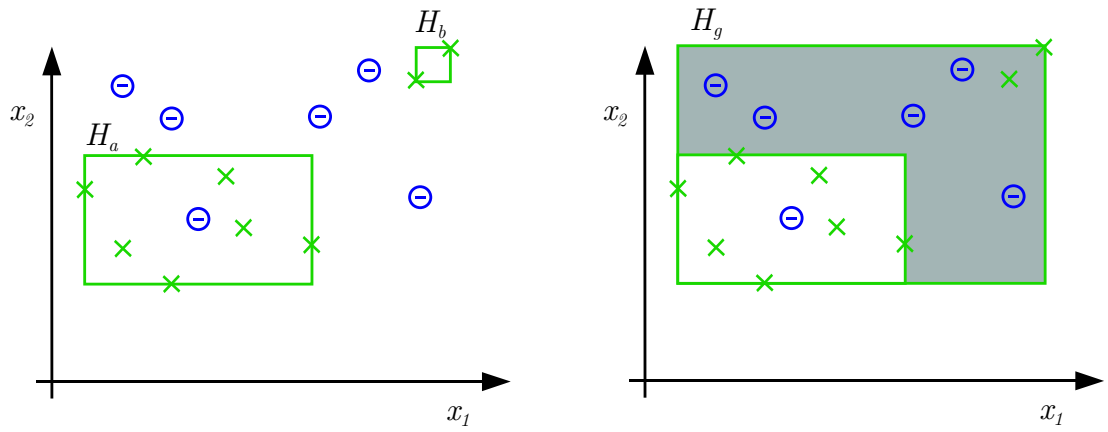


Figure 2.2: The doubled merge test. The additionally covered input space – from the viewpoint of the bigger cluster a – is shown as the grey area on the right.

2.2.3 Prediction Mechanism

A prediction of the output value given an input position is made by assigning it the output value of the closest cluster in the input space. This procedure behaves like a rule based system if the input position is inside of a cuboid. If no rule is active, i.e. if the input position is outside of any cuboid, this procedure acts like a *nearest neighbor* approach with the difference, that not distances between an input position and learn data tuples are evaluated but the distances are determined from the input position to the cuboids. Thus the clusters can be viewed as *generalized* data tuples.

2.2.4 Pseudo-Code

The flowchart 2.3 illustrates the simplified version of the PNC 2 and algorithm 1 presents its pseudo-code. It is assumed, that the possible symbols of the output are encoded as whole numbers starting from 1. The operator "." denotes the access to an element of a *struct*. For example for a cluster represented by the tuple $(\mathbf{l}, \mathbf{r}, c, p, n)$ the element c is accessed by the term $a.c$.

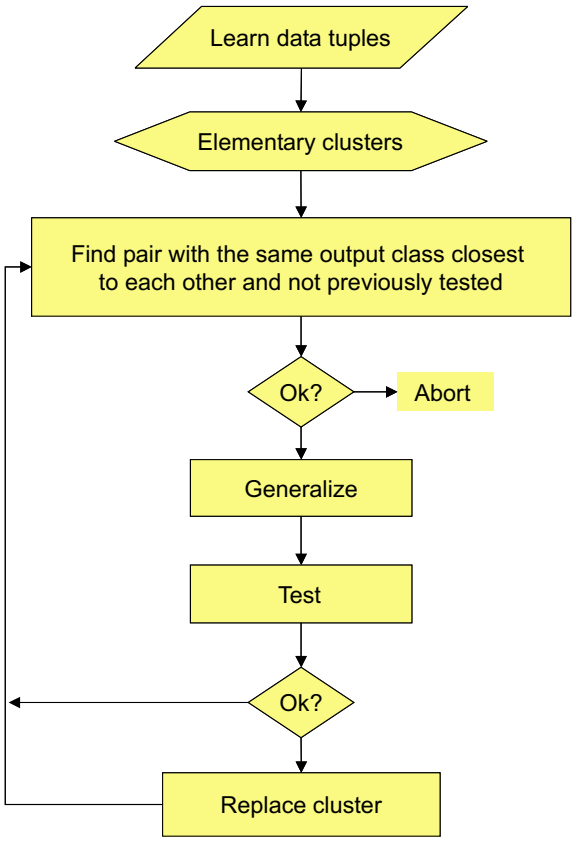


Figure 2.3: Flow chart of the simplified PNC 2 cluster algorithm.

Algorithm 1: PNC2($\mathcal{P}$)

comment: Learn clusters $\mathcal{C}$ from given data tuples $\mathcal{P}$. The global p_{min} is the minimum cluster mass and global η is the maximal percentage of negative examples relative to the cluster's mass.

global p_{min}, η

main
for each *data tuple* $P_i = (\mathbf{x}_i, y_i)$
do $C_i = \text{CREATE}(\mathbf{x}_i, y_i)$ **comment:** create elementary clusters

for each *output class* s
do
take the pair of clusters a *and* b *with output class* s *which are closest to each other and have not been taken before*
continue loop if no pair is found
$g = \text{GENERALIZE}(a, b)$
$g.n = \text{COUNTNEGATIVEEXAMPLES}(g, \mathcal{P})$
if $\text{MERGETEST}(g.n, a, b)$
then *replace the clusters* a *and* b *by* g

for each *cluster* C_t
do **if** $C_t.p \leq p_{min}$ **comment:** if cluster mass is below threshold
then *delete cluster* t

return $(\mathcal{C})$

Algorithm 1: PNC2($\mathcal{P}$)

procedure CREATE($\mathbf{x}, y$)
comment: create new elementary cluster a from given data tuple $P = (\mathbf{x}, y)$
$a.\mathbf{l} = \mathbf{x}$
$a.\mathbf{r} = \mathbf{x}$
$a.c = y$
$a.p = 1$
$a.n = 0$
return (a)

procedure GENERALIZE(a, b)
comment: create new cluster g as generalization of the two given clusters a and b
$g.\mathbf{l} = min(a.\mathbf{l}, b.\mathbf{l})$
$g.\mathbf{r} = max(a.\mathbf{r}, b.\mathbf{r})$ **comment:** component wise minima and maxima
$g.p = a.p + b.p$
$g.c = a.c$
return (g)

procedure DISTANCE(a, b)
comment: calculate distance d of clusters a and b to each other
$d = 0$
for each *input* j
do
if $a.l_j > b.r_j$
then $d = d + a.l_j - b.r_j$
else if $a.r_j < b.l_j$
then $d = d + a.l_j - b.r_j$
return (d)

procedure ISCOVERED($\mathbf{x}, a$)
comment: determine if data tuple's input vector $\mathbf{x}$ is covered by cluster a
$b =$ CREATE($\mathbf{x}, 0$) **comment:** output class is arbitrary
return (DISTANCE(a, b) == 0)

procedure COUNTNEGATIVEEXAMPLES($a, \mathcal{P}$)
comment: count number n of negative examples covered by cluster a
$n = 0$
for each *data tuple* $P_i = (\mathbf{x}_i, y_i)$
do
if $y_i \neq a.c$ **and** ISCOVERED($\mathbf{x}_i, a$)
then $n = n + 1$
return (n)

procedure MERGETEST(n, a, b)
comment: decide upon the number of positive and negative examples if cluster is a valid rule
$n_{max} = min(a.p\ \eta + b.n, b.p\ \eta + a.n)$
return ($n < n_{max}$)

procedure PREDICT($\mathbf{x}, \mathcal{C}$)
comment: predict output class $\widehat{y}$ given the input vector $\mathbf{x}$
find cluster b *closest to the input vector* $\mathbf{x}$
$\widehat{y} = b.c$
return ($\widehat{y}$)

2.3 Details and Extensions

2.3.1 Distance Measure and Representation for Nominal Inputs

The simplified version is only capable of handling continuous inputs. Thus the representation is changed for the case of nominal inputs. For a nominal input j the cluster's cuboid is defined not by a lower left and an upper right bound but by a so called *bit string*, that encodes, which symbols are covered by the cuboid with respect to the input j. The bit string has as many elements as the nominal input j can take on different symbols. Each element corresponds to one of the possible symbols and can either have the value 0 or the value 1, whereas the latter means, that the symbol is covered by the bit string. It is assumed, that the symbols of a nominal input are encoded as whole numbers starting from 1. For example the three symbols a, b and c of a nominal input may be encoded as the values 1, 2 and 3. When generalizing two cuboids, the bit string for a nominal input is build by an OR-operation such that a symbol is covered by the generalized bit string, if it is covered by any of the two bit strings which are generalized.

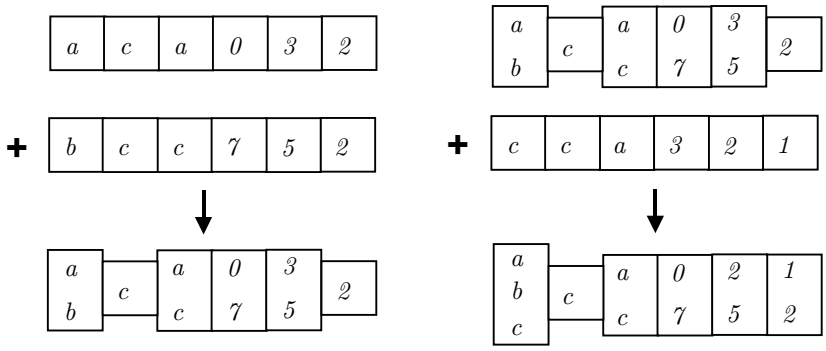


Figure 2.4: Two examples to illustrate representation and generalization. The first three inputs are nominal and can take on the symbols a, b and c whereas the latter three inputs are continuous.

The distance between data tuples resp. cuboids to each other is measured with respect to eqn. (1.29) by a *Heterogenous Normalized and Weighted Minkowski Overlap Metric* as

$$d = \sqrt[\rho]{\sum_{j=1}^{m} \omega_j \left| \frac{d_j}{\delta_j} \right|^{\rho}} \quad . \tag{2.15}$$

For continuous inputs the component wise distances d_j are determined according to eqn. 2.2 and for nominal inputs by

$$d_j = \begin{cases} 1 & \text{Symbol } x_j \text{ is activated in bitstring } \mathbf{b}_j \\ 0 & \text{else} \end{cases} \quad . \tag{2.16}$$

2.3.2 COD Problem

As described in section 1.5 the COD problem can have different, mostly negative, effects on an algorithm's performance or needs. This sections discusses how an increasingly high number of inputs affects the PNC 2 cluster algorithm and how to cope with this problem.

First of all, the so called *negative zone* is defined as that zone of the input space, that is covered by a rule's premise but inside whose the underlying true function $f_{tf}(\mathbf{x})$ has a different output value than the rule's conclusion suggests. A rule should not cover a negative zone. However, as in general $f_{tf}(\mathbf{x})$ is unknown, this cannot be guaranteed. Instead the number of negative examples of a rule is evaluated during the merge test and it is assumed, that, if a rule covers a negative zone, then this rule will have negative examples. Exactly this assumption may be a crucial problem as the following example illustrates.

Consider the scenario with two output classes in a two dimensional input space drafted in figure 2.5. The underlying true function may be defined by the surrounding cuboids. Two random samples of approx. 50 (left) and approx. 15 (right) data tuples are drawn. Now the task is to use the PNC 2 cluster algorithm to learn a model from both of the samples. As the underlying true function is known, it is possible to evaluate

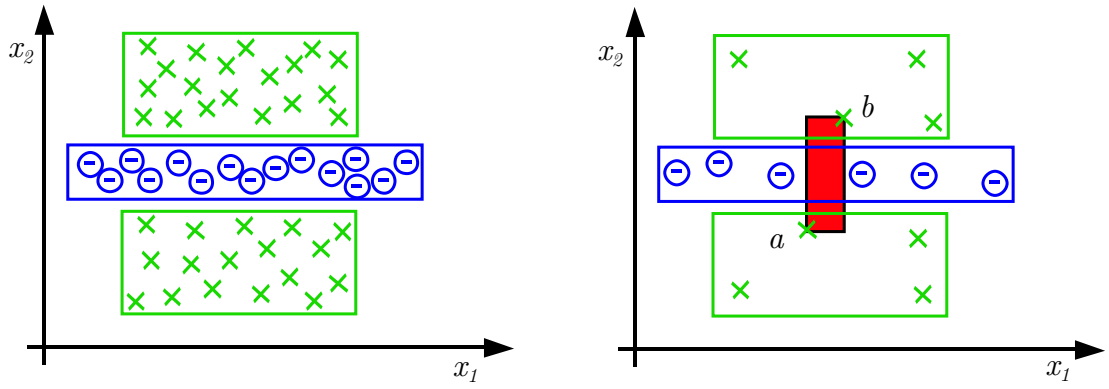


Figure 2.5: Illustration in the context of the COD problem.

the generated clusters according to how good the underlying true function is met. The arising problem gets obvious in the latter case of the small sample: The two input vectors of the data tuples a and b are closer to each other than any other data tuples with the same output class. Thus these two data tuples will be chosen for generalization and, as there are no negative examples inside the generalized cuboid, pass the merge test. However this cluster covers a negative zone.

The merge test yields the desired result only if learn data tuples lie with sufficiently high probability inside any negative zone, that may be covered by the rule. It is obvious that this probability depends somehow on the volume of the premise and on the number of learn data tuples, such that this probability increases with increasing number of learn data tuples and increasing volume of the premise.

Now the COD problem affects the PNC 2 cluster algorithm such that the volume of the premises decreases with increasing number of inputs. The following experiment illustrates this effect: 100 input vectors are generated at random for a different number of inputs. The output values are irrelevant here and neglected. Now the average volume of the cuboid generalized from 10 randomly selected data tuples is plotted against the number of inputs.

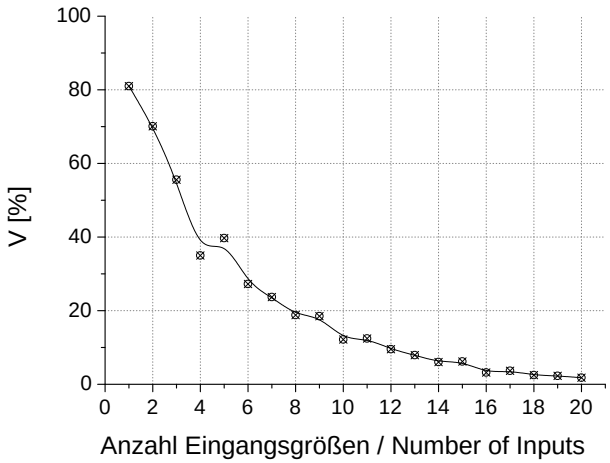


Figure 2.6: Volume of a cuboid V – normalized by the volume of the input space.

One can see, that the average volume decreases almost exponentially. Thus the chance of having a negative example in the learn data sample if a rule covers a negative zone vanishes in high dimensional spaces. This effect will be partly compensated when the clustering proceeds and the volume of the cuboids gets bigger. However, as the PNC 2 cluster algorithm works iteratively, mistakes done in the beginning can hardly be revised later on. This can easily be observed for several high dimensional benchmarks, where the learn data tuples are merged within a few clusters: The resulting rule set has a very low prediction accuracy.

Thus the mechanism how it is decided, if a data tuple is inside a cuboid, is modified during, and only during, the merge test. Up to now a data tuple is considered to be outside a cuboid, if a single component of the data tuple's input vector lies outside. Now the sum of the feature weights of the components outside is calculated

and a data tuple is still considered to be inside, if this sum does not exceed a pre-specified threshold ω_{COD}.

$$\sum \omega_j \, sgn(d_j) > \omega_{COD} \tag{2.17}$$

This approach extends the input space, that is covered by a cuboid, during the merge text. This covered input space is shown in figure 2.7 for $\omega_{COD} = 1$ for a task with two inputs.

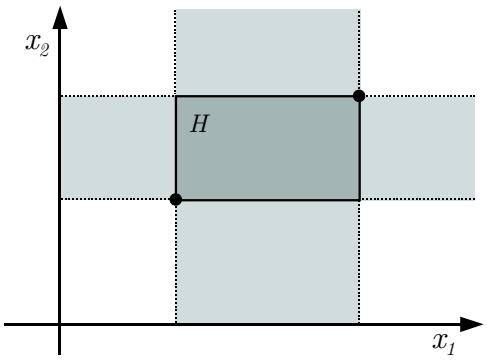


Figure 2.7: Extended cuboid (grey) during the merge test. Both inputs have a feature weight of $\omega_1 = \omega_2 = 1$.

2.3.3 How to Chose the Cuboids

All combinations of two clusters with the same output value may be generalized with each other. According to this, the clusters are assorted in groups with the same output value. These groups are processed successively. For each group iteratively always the two clusters closest to each other and not previously tested are chosen. The distances of the clusters to each other are stored in an *adjacency matrix*. This is a lower triangular matrix of the dimension $K-1$, where K is the actual number of clusters in the group. The adjacency matrix is reduced by one dimension, if two clusters are merged. The distances of the new generalized cluster to the other existing clusters must be recalculated. Note that a generalized cluster can again be merged with any other cluster – except if the parameter η is chosen to 0. In the latter case the generalized cluster does not need to be tested with any cluster that has been previously tested with one of the two clusters merged.

A problem is the size of the adjacency matrices, that increases quadratically with the number of learn data tuples. A *divide & conquer* approach is used to be able to use the PNC 2 cluster algorithm for learning tasks with a higher number of learn data tuples: To be concrete, when initializing, i.e. when transforming each learn data tuple to an elementary cluster and building up the groups of clusters with the same output value, the number of clusters within one group is bound to a maximum of $N_{G\,Max}$. If the group's size would exceed this threshold, the clusters are split in two or more approximately equally sized *sub-groups*. These sub-groups are processed independently. Afterwards it is tried to further merge the generated clusters of the different sub-groups with each other.

2.3.4 Context Sensitive Feature Selection

The feature selection methods described in section 1.6 decide globally upon the relevance or irrelevance of an input. However, sometimes an input is only relevant in a special context, i.e. in a special part of the input space. Such an input should not be eliminated. Motivated by a similar approach for the RISE algorithm (described in section 2.4.2), the following *context sensitive feature selection* (CSFS) is defined for the PNC 2 cluster algorithm.

After the clustering process is finished, each cluster C is processed independently. The single components of the cluster's cuboid are sorted in decreasing order according to the so called *degree of coverage* that is defined as

$$\xi_j = \begin{cases} \dfrac{|r_j - l_j|}{x_{range\,j}} & \textit{continuous inputs} \\ \dfrac{\sum_{w=1}^{S_j} b_{j\,w}}{S_j} & \textit{nominal inputs} \end{cases} . \tag{2.18}$$

For continuous inputs the degree of coverage is determined as the range of the cuboid component relative to the range of the input. For nominal inputs it is evaluated how many percent of the number of possible symbols S_j are covered by the corresponding bit string.

Now the single components of the cluster's cuboid are processed iteratively in order of decreasing degree of coverage and it is tried to maximally generalize the component under consideration. To be concrete, for a nominal input the bit string is set to be **1**, i.e. now covers every possible symbol. For continuous inputs first the left and then the right bound are set to $l_j = -\infty$ resp. $r_j = \infty$. The generalization is accepted if it does not increase the number of negative examples – determined the same way as during the merge test – of the cluster. Note that this method works iteratively, i.e. in each step always the cluster is used as it emerged from the previous generalization tries.

Before the CSFS is applied each cuboid has – with respect to the memory needed to store it – a size of m. Afterwards the size of a cuboid is determined as

$$\sum_{j=1}^{m} h_j \ , \tag{2.19}$$

whereas h_j is determined for continuous inputs as

$$h_j = \begin{cases} 1 & l_j \neq -\infty \wedge r_j \neq \infty \\ 0 & l_j = -\infty \wedge r_j = \infty \\ 0.5 & else \end{cases} \tag{2.20}$$

and for nominal inputs as

$$h_j = \begin{cases} 0 & \mathbf{b}_j = \mathbf{1} \\ 1 & else \end{cases} \ . \tag{2.21}$$

The time complexity of this context sensitive feature selection method is $O(m^2 N_L K)$, whereas N_L denotes the number of learn data tuples, m denotes the number of inputs and K is the number of clusters. The approach scales linearly with the number of clusters, as each cluster is processed independently from the others. For each cluster iteratively each input is processed and a merge test needs to be done. This in turn scales linearly with the number of inputs and the number of learn data tuples.

2.3.5 k-Nearest-Cluster Prediction Mechanism

Classification Tasks The prediction mechanism described in the basic version of the PNC 2 cluster algorithm is a kind of an 1-nearest-neighbor prediction, as the generated clusters can be seen as generalized data tuples. Many studies have shown, that a *k-nearest-neighbor* prediction can improve the prediction accuracy. For classification tasks it is possible to realize a similar approach for the PNC 2 cluster algorithm. However it is preferable not to use a fixed number of nearest clusters, but to adaptively adjust the neighborhood for the given input position. Otherwise the high compression rates of the PNC 2 cluster algorithm could lead to the fatal effect described in the following hypothetical scenario with the two possible output classes a and b: The learn data tuples were merged into 5 different clusters. One of them has an output value of a, whereas all the other ones have an output value of b. An evaluation of the 3 nearest clusters would always lead to a prediction of the output class b irrespective of the input position.

The neighborhood is adapted individually for the given input position $\mathbf{x}$ as follows: First of all, the distance d_{min} of the cluster nearest to the input position is determined. The neighborhood distance is now defined as

$$d_{ref} = w_{Kernel}\, d_{min} + w_{Kernel\,Min} \ , \tag{2.22}$$

whereas w_{Kernel} and $w_{Kernel\,Min}$ are adjustable parameters. A setting of $w_{Kernel} = 0$ and $w_{Kernel\,Min} = 0$ would lead to the original *1-nearest-cluster* prediction. The parameter $w_{Kernel\,Min}$ can be transparently adjusted by coupling it to the average diameter of the cuboids, that is calculated according to equation (2.26).

The output values of all clusters with a distance less than d_{ref} to the given input position $\mathbf{x}$ are evaluated by a distance weighted voting. It is determined for each possible output symbol s how much it is suggested by the clusters in the neighborhood by evaluating

$$\mu_s = \sum_{t=1}^{K} e \left(1 - \frac{d_t - d_{min}}{d_{ref} - d_{min}}\right) \qquad \text{with} \qquad e = \begin{cases} 1 & d_t < d_{ref} \wedge s = c_t \\ 0 & else \end{cases}, \tag{2.23}$$

whereas K denotes the number of clusters and d_t resp. c_t is the distance resp. the output value of the t-th cluster. The most suggested output value

$$\widehat{y} = \arg\max_{s=1}^{S_y}(\mu_s) \tag{2.24}$$

is predicted. Thus the impact of a cluster on the predicted output value is maximal for a cluster with a distance of d_{min} and it vanishes for a cluster at the boundary of the neighborhood.

Regression Tasks The PNC 2 cluster algorithm can be easily extended to handle regression tasks. Therefor the continuous output is discretized by a suitable discretization method, as described in section 1.8.2. We use equidistant binning for this purpose, because it is a simple and fast method that yields a uniform splitting of the range of possible output values. The discretized output values can then be treated as different symbolic values and the algorithm can be used as ordinary. To compensate partly the loss of information, that results from the discretization, the output value of each cluster is re-calculated after the clustering is finished as the mean of the output values of all the data tuples merged within.

The clusters are transformed to a fuzzy rule set[2]. Multiplication and ordinary sum are used as fuzzy AND- resp. OR-operator and defuzzification is done using eqn. (1.8). The rules take the same form as in the basic version according to eqn. (2.3), except that the membership degree of the premise is now evaluated by

$$\mu = \exp - \left(\frac{d}{\sigma_{msf}}\right)^{\varsigma}. \tag{2.25}$$

The parameters ς and σ_{msf} control the width and the form of the fuzzy membership functions. Common values for ς are 1 and 2. Note that the parameter σ_{msf} is set globally for all rules. To transparently adjust this parameter it is coupled to the average cuboid diameter, that is determined using the distance function (2.15) whereas the component wise distances are calculated as

$$d_j = \begin{cases} r_j - l_j & \textit{continuous inputs} \\ \sum_{w=1}^{S_j} b_{w\,j} & \\ \hline S_j & \textit{nominal inputs} \end{cases}. \tag{2.26}$$

S_j denotes the number of possible symbols of a nominal input j and $\sum_{w=1}^{S_j} b_{w\,j}$ is the number of symbols of the input j that is covered by the bit string $\mathbf{b}_j$.

If the metric parameter ρ equals ς, it is possible to project the membership functions to the single inputs and to get the following, more familiar rule

$$\text{IF} \quad x_1 = \mu_1 \quad \text{AND} \quad x_2 = \mu_2 \quad \text{AND} \quad \ldots \quad \text{AND} \quad x_m = \mu_m \quad \text{THEN} \; y = c \tag{2.27}$$

with

$$\mu_j = \exp - \omega_j \left(\frac{1}{\sigma_{msf}} \frac{d_j}{\delta_j}\right)^{\rho}. \tag{2.28}$$

Note that the representation of the generated clusters as fuzzy rules is a rather unfavorable decision, as, for continuous inputs, each single rule needs its own membership functions. This could be alleviated by transforming continuous inputs to nominal ones in a preprocessing step – either by an automatic discretization method or manually by defining membership functions as usual.

[2]To clarify things: The cluster's hitrate is not re-calculated for the fuzzy rule, it is just copied. All previously described mechanisms as the context sensitive feature selection etc. are applied before.

2.3.6 Time Complexity

The time complexity of an algorithm is of great importance to decide upon its feasibility for a particular learning task. It specifies how the runtime of the algorithm increases with increasing size of the learning task, which is – in the context of this work – given by the number of learn data tuples N_L and by the number of inputs m.

The merge test has a time complexity of $O(N_L\, m)$, as for each data tuple with a different output value it has to be checked, if the data tuple is inside or outside of the generalized cuboid and this in turn scales linearly with the number of inputs. The *worst case*[3] number of merge tests done, results from the combination of all possible pairs of two clusters with the same output value. Thus the whole algorithm has a worst case time complexity of $O(N_L^3\, m)$. But, as shown in experimentell tests with several benchmarks in section 4.5, the average runtime increases only quadratically with the number of learn data tuples.

2.3.7 Pseudo–Code

This sections presents the pseudo-code of the PNC 2 cluster algorithm for the case of heterogenous continuous and nominal inputs as well as the improved prediction mechanism for classification tasks. The routines for the case of regression tasks, for the use of a hitrate to rate the rules and for the efficient management of the adjacency matrices are omitted.

A bit string is used to represent the cuboid for nominal inputs. This bit string indicates, which of the possible symbols of a nominal input are covered by the cuboid. It is assumed, that the possible symbols are encoded as whole numbers starting from 1. A heterogenous weighted and normalized Minkowski overlap metric according to eqn. 2.15 is used. See section 1.8.1 and 1.8.2 for details on how to calculate normalization factors and feature weights.

Algorithm 2: PNC2($\mathcal{P}$)

---

comment: The globals $\boldsymbol{\omega}$ and $\boldsymbol{\delta}$ contain feature weights and normalization factors for each input. The global ρ defines the Minkowski metric used and the globals w_{Kernel} and $w_{Kernel\,Min}$ control the neighborhood for the *nearest cluster* prediction. The program **main** and the procedures COUNTNEGATIVEEXAMPLES and MERGETEST are omitted here as they are same as in algorithm 1.

global $\boldsymbol{\omega}, \boldsymbol{\delta}, \rho, w_{Kernel}, w_{Kernel\,Min}, \omega_{COD}$

procedure CREATE(P)
comment: create new elementary cluster a from given data tuple $P = (\mathbf{x}, y)$
for each *input* j
do { **if** *input* j *is nominal*
then *set bit string* $a.\mathbf{b}_j$ *to cover symbol* x_j
else { $a.l_j = x_j$; $a.r_j = x_j$ }

$a.c = y$
$a.p = 1$
$a.n = 0$
return (a)

procedure GENERALIZE(a, b)
comment: create new cluster g as generalization of the two given clusters a and b
for each *input* j
do { **if** *input* j *is nominal*
then *set bit string* $g.\mathbf{b}_j$ *to cover all symbols covered by bit strings* $a.\mathbf{b}_j$ *or* $b.\mathbf{b}_j$
else { $g.l_j = min(a.l_j, b.l_j)$; $g.r_j = max(a.r_j, b.r_j)$ }

$g.p = a.p + b.p$
$g.c = a.c$
return (g)

---

[3] In the worst case each merge test fails and thus the cluster population at the end of the clustering process is the same as directly after initialization. This case will rarely occur for practical learning tasks.

Algorithm 2: PNC2($\mathcal{P}$)

procedure DISTANCE(a, b)
comment: calculate distance d of the two clusters a and b to each other
for each *input* j
do **if** *input* j *is nominal*
then *set* d_j *to* 1 *if at least one bit is set in* **both** *bit strings* $a.\mathbf{b}_j$ *and* $b.\mathbf{b}_j$
else **if** $a.l_j > b.r_j$
then $d_j = d + a.l_j - b.r_j$
else if $a.r_j < b.l_j$
then $d_j = d + b.l_j - a.r_j$

$$d = \sqrt[\rho]{\sum \omega_j \left(\frac{d_j}{\delta_j}\right)^{\rho}}$$

return (d)

procedure ISCOVERED($\mathbf{x}, a$)
comment: determine if data tuple's input vector $\mathbf{x}$ is covered by cluster a
$\omega_{sum} = 0$
for each *input* j
do **if** *input* j *is nominal*
then $\omega_{sum} = \omega_{sum} + \omega_j$ *if symbol* x_j *is* **not** *covered by bit string* $a.\mathbf{b}_j$
else **if** $x_j < a.l$ **or** $x_j > a.r$
then $\omega_{sum} = \omega_{sum} + \omega_j$

return ($\omega_{sum} < \omega_{COD}$)

procedure PREDICT($\mathbf{x}, \mathcal{C}$)
comment: predict output class $\widehat{y}$ given the input data vector $\mathbf{x}$
for each *cluster* C_t
do $d_t = distance\ of\ input\ vector\ \mathbf{x}\ to\ cluster\ C_t$
$d_{min} = \min_{t=1}^{K}(d_t)$
$d_{ref} = d_{min}\, w_{Kernel} + w_{Kernel\,Min}$

$\boldsymbol{\mu} = \mathbf{0}$
for each *cluster* C_t
do **if** $d_t < \delta_{ref}$
then $s = C_t.c$
$\mu_s = \mu_w + 1 - \frac{d_t - d_{min}}{d_{ref} - d_{min}}$

$\widehat{y} = \arg\max_{s=1}^{S_y}(\mu_s)$ **comment:** predict most suggested output class
return (y)

2.4 Related Work

This section compares the PNC 2 cluster algorithm with the most similar existing algorithms, namely the NGE and the RISE algorithm.

2.4.1 The NGE Algorithm

The *Nearest Generalized Exemplar* (NGE) algorithm [Sal91, Wet94, Aha95a] was developed by SALZBERG. The representation uses so called *exemplars*, that consist of an axis-parallel cuboid in the input space and an associated output value. Essentially, as described for the PNC 2 cluster algorithm in section 2.2.1, data tuples can be transformed to trivial exemplars, two exemplars can be generalized or distances between data tuples and exemplars can be calculated. The output value of the nearest exemplar is used to make a prediction given an input position. According to this quick overview, representation and prediction mechanism are essentially the same as in the basic version of the PNC 2 cluster algorithm.

However the search, i.e. the learning, is fundamentally different. The NGE algorithm gets initialized by transforming a pre-specified number of learn data tuples to trivial exemplars. Incrementally the remaining data tuples are processed as follows. A data tuple is transformed into a trivial exemplar E_t and the closest exemplar E_p in the actual population is determined. The trivial exemplar E_t is merged with E_p, if the output value

of these two exemplars coincides. Else the second nearest exemplar of the actual population is tried. If again the output value is different, the trivial exemplar is inserted into the population. Therewith a learning step is finished and the next remaining learn data tuple is processed.

The main difference, compared with the PNC 2 cluster algorithm, is the incremental learning process, that leads to an result, that depends on the order, in which the learn data tuples are processed and that allows the merging of exemplars that cover negative examples.

Several variants of the original NGE algorithm are described in [WD95]. Of special interest in the context of this work are two variants, that transform the originally incremental algorithm into a *batch* version. This is the *Overlapping Batch* NGE (OBNGE) and the *Batch* NGE (BNGE) algorithm. In the first algorithm, two exemplars are merged, if the generalized exemplar does not cover any negative examples. In the latter one, two exemplars are merged, if the generalized exemplar does not overlap with any existing exemplar. Thus – especially for the OBNGE – the approach is quite similar to the PNC 2 cluster algorithm. However the OBNGE algorithm does not have any mechanism to circumvent the COD problem and thus achieves only poor predictions accuracies for most tasks. The PNC 2 cluster algorithm is compared experimentally on several benchmarks to the NGE algorithm and its variants in section 4.7.1. The results show, that the PNC 2 generates substantially better and much smaller models than any NGE algorithm.

2.4.2 The RISE algorithm

The *Rule Induction from a Set of Exemplars* (RISE) algorithm [Dom95, Dom96, Dom97] was developed by DOMINGOS and uses the same representation and a similar prediction mechanism as the NGE or the PNC 2 cluster algorithm. Different however is the learning procedure which is described in the following: First of all, each learn data tuple is – analogous to the PNC 2 Algorithmus – considered to be a single rule. Afterwards the set of all rules is iterated several times until an abortion criterion is met. In doing so, each rule is taken and and it is tried to generalize it to cover another positive example. The generalization is accepted, if it does not have a negative effect on the prediction accuracy – determined by a leave-one-out cross-validation within the learn data sample – of the whole rule set. Note, that not data tuples are merged with each other like in the PNC 2 cluster algorithm. Rules are extended to cover another data tuple and thus identical rules can emerge. These have to be detected and deleted.

The main difference compared with the PNC 2 cluster algorithm is the search strategy used. The PNC 2 cluster algorithm uses a local decision criterion and continues as long, as two clusters could potentially be merged. Insofar the search strategy of the PNC 2 cluster algorithm can be viewed as *global*. The RISE algorithm, on the other hand, uses a global decision criterion – the prediction accuracy of the whole rule set – and considers only the data tuples nearest to a rule for generalization. This search strategy can be viewed as *local*.

DOMINGOS describes two approaches how to reduce the number of rules in a post-processing step. First, all rules are deleted that does not *win* at least one learn data tuple. A rule wins a data tuple, if it is the nearest one. If two or more rules have the same distance – which may happen quite often as the rules may overlap – the rule with the highest laplace corrected hitrate wins. In [Dom97] it is shown, that this approach leads to an average reduction of approximately 90%. However this has a small negative impact on prediction accuracy. The other approach aims at reducing the number of inputs accessed in the premises of the rules. Within each rule an input is deleted, if it does not separate any negative example, i.e. if each negative example of the rule is inside the premise for this input under consideration. This approach is similar to the context sensitive feature selection of the PNC 2 cluster algorithm, but, in contrary, achieves – according to DOMINGOS – only poor compression rates.

The PNC 2 cluster algorithm is compared experimentally with the RISE algorithm in section 4.7.2. The results show, that the PNC 2 mostly achieves better prediction accuracies using much smaller models.

Chapter 3

Validation and Parameter Tuning

This chapter gives an introduction to different loss functions to evaluate the prediction accuracy of a learned model or of the learning algorithm itself. Two methods – namely the N-fold cross-validation and the N-fold repetition – of how to estimate the loss function value of a learning algorithm for a given data sample are described. A problem arises if the algorithm has free tunable parameters. These parameters must be adjusted systematically – otherwise the results can be corrupted. Thus, based on a work of SALZBERG, the approach of the *tough validation* is defined.

3.1 Model Evaluation

3.1.1 Prediction Accuracy

The most important criterion to evaluate a learned model is its prediction accuracy. Some commonly used loss functions for classification and for regression tasks are described in the following paragraph. Usually these loss functions are evaluated with respect to a new and unseen test data sample. Therefor, based upon the particular input vectors, a prediction $\widehat{y}$ of the output value is done for each test data tuple. Then the difference between the real and the predicted output value is evaluated and summarized in a single loss function value. It is also possible, to evaluate the prediction accuracy with respect to the learn data sample. However, this seldom yields any useful information – especially if IBL algorithms are used, as these approaches usually lead to perfect or almost perfect prediction accuracy on the learn data sample. To prevent misunderstandings: If not stated otherwise, the term *prediction accuracy* always refers to a separate test data sample.

Classification The *mean classification error* (MCE) specifies the percentage of output values, that were misclassified.

$$MCE = \frac{\sum_{i=1}^{N} e_i}{N} \quad \text{with} \quad e_i = \begin{cases} 0 & \widehat{y}_i = y_i \\ 1 & \widehat{y}_i \neq y_i \end{cases} \tag{3.1}$$

Thereby $\widehat{y}_i$ denotes the predicted and y_i the real output value of the i-th data tuple and N denotes the sample size. The resulting loss function values are within the interval [0..1] and are directly interpretable to a human.

One can build up the so called *confusion matrix* $\mathbf{O}$ to get a deeper insight into the types of the misclassifications. This matrix summarizes how often a particular output class is misclassified as an other particular one. The confusion matrix is a quadratic matrix, whose size equals the number of possible output classes S_y. The element $o_{w,z}$, counts, how often the class z was misclassified as class w.

$$o_{w,z} = \sum_{i=1}^{N} e_i \quad \text{with} \quad e_i = \begin{cases} 1 & \widehat{y}_i = w \ \wedge \ y_i = z \\ 0 & \text{else} \end{cases} \tag{3.2}$$

Regression The *mean absolute* and the *mean square error* (MAE resp. MSE) indicate the absolute resp. squared error, that is done on average about all data tuples of the sample.

$$MAE = \frac{\sum_{i=1}^{N} |\widehat{y}_i - y_i|}{N} \tag{3.3}$$

$$MSE = \frac{\sum_{i=1}^{N} (\widehat{y}_i - y_i)^2}{N} \tag{3.4}$$

Again, $\widehat{y}_i$ denotes the predicted and y_i the real output value of the i-th data tuple. N is the sample size. Usually the resulting loss function values are not directly interpretable. However the value can be normalized by the error of the so called *baseline* prediction [RNH+96], i.e. by the error, that would have been done, if a fixed output value y_{const} would have been predicted for each data tuple irrespective of the input vector. The median resp. the mean of the output values of the data sample is used as value y_{const} for the MAE resp. MSE loss function[1]. The normalized loss function values should be far less than 1. Otherwise one should have serious doubts about the prediction capabilities of the learned model, as a simple baseline prediction yields always the same prediction accuracy.

3.1.2 Methods to Estimate the Prediction Accuracy

The prediction accuracy of different learning algorithms can be compared experimentally with each other with respect to a given learning task, i.e. with respect to a given data sample. Below, first the concept of the *true* loss function value Q_P^* of a model is introduced and thereupon the *true* loss function value of a learning algorithm Q^* is defined. Then the N_R-fold cross-validation and the N_R-fold repetition are introduced as possible estimators of Q^*.

The true loss Q_P^* of a model with respect to a given learning task is the resulting loss if an infinite test data sample would be evaluated. Thereby the occurring input vectors would be drawn according to the true distribution $\mu_{tf}(\mathbf{x})$ and the corresponding output value would obey to the underlying true function $y = f_{tf}(\mathbf{x})$. Both, true function and distribution, are naturally unknown and thus the true loss cannot be determined exactly. Usually only a limited amount of test data tuples is available and it is only possible to determine the loss $\widehat{Q}_P$ with respect to this limited sample.

The true loss of a learning algorithm with respect to a given learning task for a fixed learning sample size is the resulting average of the model's true loss function values Q_P^*, if infinitely often a model is learned using a learn sample of size N. In general Q_P^* cannot be determined exactly too. Thus the N_R-fold cross-validation, the N_R-fold repetition or similar approaches are used as estimator.

- **N_R-fold cross-validation:** The data sample is divided into N_R sub-samples of approximately the same size. Each of the smaller sub-samples is kept aside as test data sample once and the data tuples of the other $N_R - 1$ sub-samples are used to learn a model whose prediction accuracy $\widehat{Q}_P$ is then determined on the test data sample. Kohavi reports in [Koh95], that $N_R = 10$ is a good setting, even if available computational resources would allow a bigger value. A special type of cross-validation is the so called *leave-one-out cross-validation* (LOOCV) for which N_R is chosen as the number of data samples.
- **N_R-fold repetition:** The data sample is randomly divided into a learn and a test data sample of the pre-specified sizes N_R times. A model is learned using the learn data sample and its prediction accuracy $\widehat{Q}_P$ is evaluated with respect to the test data sample.

The mean of the N_R single prediction accuracies $\widehat{Q}_P$ is reported as result $\widehat{Q}$. It is possible to calculate the standard error of this mean as

$$\sigma_{\widehat{Q}}^* = \frac{\sigma_Q}{\sqrt{N_R}}. \tag{3.5}$$

[1] For the MSE loss function this is equivalent to a normalization to the deviation of the output values.

Thereby σ_Q denotes the standard deviation of the prediction accuracies $\widehat{Q}_P$. The estimated value differs with a probability of approximately 67% not more than σ_Q^* from the real value – provided that the prediction accuracies $\widehat{Q}_P$ are normally distributed[2].

3.1.3 Other Model Evaluation Criterions

Apart from the prediction accuracy there are some other criterions, that are used to compare learned models with each other.

- *Size.* The size of a model is usually given as the memory that is needed to store the learned model. For a rule based model this is the number of terms in the premise and conclusion summed up over all rules. Among other things, the size is important to evaluate the interpretability of the model.
- *Runtime.* The computational costs to first learn a model from the learn data sample and second to make predictions for new unknown input vectors.
- *Required background knowledge.* The knowledge that a user must have to parameterize the learning algorithm to reach an acceptable prediction accuracy.

3.2 Parameter Tuning

Salzberg describes in [Sal99] a problem he calls *repeated tuning*:

> *Many researchers tune their algorithms repeatedly in order to make them perform optimally on at least some data sets. At the very least, when a system is being developed, the developer spends a great deal of time determining what parameters it should have and what the optimal values should be. [...] Fortunately one can perform virtually unlimited tuning without corrupting the validity of the results. The solution is to use cross-validation entirely within the training set itself. [...] When the parameters appear to be at their optimal settings, accuracy can finally be measured on the test data.*
> *Thus when doing comparative evaluations,* everything *that is done to modify or prepare the algorithms must be done in advance of seeing the test data. This point will seem obvious to many experimental researchers, but the fact is that papers are still appearing in which this methodology is not followed. In the survey by Flexer [Fle96], only 3 out of 43 experimental papers in leading neural network journals used a separate data set for parameter tuning; the remaining 40 papers either did not explain how they adjusted parameters or else did their adjustments after using the test set.*

Therewith Salzberg has appointed a common problem in existing literature. As well he has shown a solution how to avoid this problem by using cross-validation or similar approaches to automatically tune all free parameters. In the following this approach is called *tough validation* – in contrast to the normal validation usually used.

Yet, and Salzberg also partly addresses this, there are lot of situations, where a normal validation is quite useful. The accomplishment of a tough validation requires much more work and computational power. Default parameter sets must be defined for the algorithm's free parameters and a parameter tuning, that automatically chooses a suitable parameter set based upon some cross-validation results, must be realized. These are non-trivial tasks. When developing a new algorithm, first of all, one is interested in quickly evaluating and – if successful – demonstrating its capabilities. The easiest way to do this is to implement a first prototype of the algorithm and to manually tune free parameters or exchange whole parts and strategies – while looking at the algorithms prediction accuracy on some chosen benchmarks. A tough validation is not executable in this stage of algorithm design at justifiable costs.

3.2.1 Tough Validation

The tough validation described here is derived from Salzberg's recommended approach in [Sal97]. Firstly all free parameters of the algorithm are considered be be tuned automatically and it is assumed, that so called

[2] Note that this may be an overly optimistically estimation as it is assumed in eqn. (3.5), that the N_R single values are obtained from statistically independent experiments. However this is not given if data tuples are re-used in the learn and test data samples several times.

default parameter sets have been defined for them. The following subsection then discusses how to reduce the number of free parameters by choosing, based upon a preliminary study, a fixed setting for some parameters.

The given data sample is divided into several learn and test data samples using a N_R-fold repetition or cross-validation. A so called *parameter tuning* is executed for each of the learn data samples to automatically select a suitable parameter set. The selected set is then used to learn a model with the learn data sample and the model's accuracy is determined with respect to the test data sample. The mean of the single N_R prediction accuracies is reported.

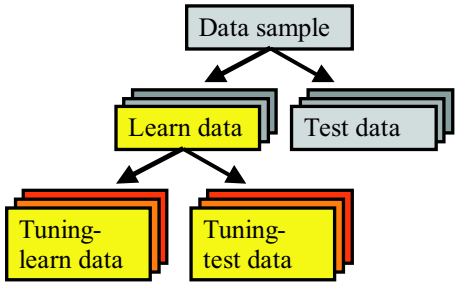


Figure 3.1: Data splitting for tough validation using N-fold repetition.

The parameter tuning is done as follows: The particular learn data sample is further sub-divided into N_{Tune} tuning learn and tuning test data samples using a N_{Tune}-fold repetition or cross-validation. A model is learned for each possible parameter set for each of the tuning learn data samples and the model's prediction accuracy is determined for the corresponding test data samples. A supposable good parameter set is selected based upon these prediction accuracies considering some additional criterions like the size of the learned models. To simple selection strategies are:

- *Mean learn sample accuracy.* The mean learn sample accuracy is the mean of the single prediction accuracies for the N_{Tune} repetitions or cross-validations done with the particular learn data sample.
- *Mean learn sample rank.* The parameter sets are sorted for each of the N_{Tune} repetitions or cross-validations with respect to the prediction accuracy achieved by the models learned with them. Then each time the parameter set, that leads to the model with the best prediction accuracy gets a rank of 1 and the other sets follow with ascending rank. If two parameter sets lead to the same prediction accuracy, then both sets are assigned the same rank. Now the mean learn sample rank is the rank of the parameter sets averaged over the N_{Tune} repetitions or cross-validations done with the particular learn data sample.

Note, that there may be an over fitting problem, as the selected parameter set is only optimal with respect to the particular tuning learn and test data samples. It is not guaranteed, that it is also optimal with respect to the learn and test data samples, that are finally used to measure prediction accuracy. Thus the prediction accuracies of an algorithm achieved with a tough validation will usually be worse than the ones of a normal validation – this is why the term parameter *tuning* and not *optimization* is used. In addition a human would be far better in choosing a good parameter set than the automated parameter tuning. This is because a human is capable of more complex decision strategies and considerations.

3.2.2 Parameter Classes for Tough Validation

The approach of the tough validation described above requires that each of the default parameter sets defined is considered within the scope of the parameter tuning. Thus the computational costs increase with an increasing number of default parameter sets, which in turn usually increases with the number of free parameters. Therefore it is advisable to chose suitable fixed settings for as many parameters as possible. This can be done by in deep thoughts about how a parameter affects learning of the algorithm or by a preliminary study with some well selected *development* benchmarks. From now on, parameters defined in these two ways are called *design decisions* resp. *design parameters*. The remaining free parameters of the learning algorithm are called *strategy parameters*.

The selection strategy used for the parameter tuning and the parametrization of the cross-validation or repetition used to estimate the learn sample accuracy or rank are free parameters too. Either these parameters are defined ad hoc or suitable values are also chosen in a preliminary study. In the following these parameters are called *tuning parameters*.

At last there are the parameters, that control the experiment design itself. This is, for example, the learn and test data sample size and the number N_R of repetitions or cross-validations. These *experiment parameters* are uncritical, i.e. they cannot be defined in a preliminary study. One can find several examples of how to chose these parameters by looking at published experimental comparisons studies like [WD95, Dom97, WM00a].

Table 3.1: Parameter types for tough validation.

Type	**Explanation**
Design decisions	Free parameters of the learning algorithm that are defined by thoughts about the functionality of the particular parameter.
Design parameters	Free parameters of the learning algorithm that are defined by preliminary studies.
Strategy parameters	Free parameters of the learning algorithm that are tuned within the scope of the parameter tuning.
Tuning parameters	Parameters, that define the exact proceeding for the automated parameter tuning.
Experiment parameter	Parameters, that determine the experiment design itself.

The various parameter types, that need to be distinguished in the context of a tough validation, are summarized in table 3.1. The assignment of the free parameters of the learning algorithm to one of the three possible types is done by the developer and thus is a somehow subjective decision. The benchmarks used for the preliminary studies cannot be used for a tough validation anymore. The results of the learning algorithm for these tasks either must not be reported within the scope of a tough validation or they must be adequately marked.

Chapter 4

Experiments

This chapter first describes the preliminary thoughts and experiments done to get a deep understanding about the algorithm, to reduce the number of free parameters, to find some good standard sets for the remaining parameters and to decide upon the strategies used for the parameter tuning. Afterwards the PNC 2 cluster algorithm is compared with the most similar existing approaches in an extensive benchmark study.

4.1 Overview

A new learning algorithm should be compared with the most similar existing algorithms [Sal99]. For the PNC 2 these are the NGE and the RISE algorithm described in section 2.4 and the kNN algorithm described in section 1.3. A former study done with the predecessor of the PNC 2 cluster algorithm has been published in [Hae01b, Hae01a]. Thereby some free parameters of the algorithm have been classified as strategy parameters, that were tuned automatically within the particular learn data sample. However, the other free parameters of the algorithm, the tuning parameters and the default parameter sets for the strategy parameters have been defined ad hoc without further examination. Now for the PNC 2 cluster algorithm all free parameters are classified according to table 3.1 and are systematically studied with respect to their functionality. Then suitable settings are chosen based upon the results. Therefor preliminary thoughts and experiments with some well selected benchmarks are done. These are documented in the following two sections.

Table 4.1: Overview of preliminary studies.

Abbreviation	**Question**	**Comment**
EXPA	Design parameters	Choice of normalization and discretization method, decision on the usage of rule rating strategies etc.
EXPB	Strategy parameters	Parameter studies to find good default parameter sets
EXPC	$N_{G\,Max}$	Effect of the design decision $N_{G\,Max}$ on runtime, prediction accuracy and model size with increasing learn data sample size
EXPE	Tuning parameters	
	1: Validation method	Decision between 10-fold cross-validation and 10 or 50-fold repetition
	2: Additional constraints	Test of the additional criterions *maximal model size* and *minimal merge rate*
	3: Objective	Decision between learn sample accuracy and learn sample rank
	4: CSFS	Effect of a permanent activation of the context sensitive feature selection

Table 4.1 provides an overview of the questions studied in the various preliminary experiments. All experiments, except ExpC, were done using the benchmarks Australian, Hepatitis, Mushrooms, Seg and Temp. The benchmarks Dna, Letter and Wave were used for the experiments ExpC as larger quantities of data tuples were required. However, the obtained results were not used, as the benchmarks Dna, Letter and Wave are also used in the benchmark comparison study in section 4.7 – otherwise the conditions of a tough validation would not be fulfilled.

4.2 Preliminary Thoughts

The six free parameters of the Pnc 2 cluster algorithm listed in table 4.2 are classified as design decisions and the specified values are – if not stated otherwise – used for **all** of the following experiments and benchmark comparison studies. Below each design decision is quickly described and the thoughts, that led to the chosen values are documented.

Table 4.2: Design decisions.

ρ	ς	$N_{G\,Max}$	N_{Bins}	δ_{Symb}	$W_{Kernel\,Min}$
1	2	250	10	0.3^{-1}	0

- The parameter ρ is used for the Minkowski metric in eqn. (2.15). A setting of 1 seems to be best, as this matches the rectangular premises used by the Pnc 2 cluster algorithm. Other values were tentatively tried. The prediction accuracy on the development benchmarks was partly slightly better but partly slightly worse. There is no clearly superior setting and thus it would not be possible to adjust this parameter as a design parameter. A tuning within the parameter tuning would be possible but seems not reasonable, as the effect of this parameter on the prediction accuracy is rather small.
- The parameter ς is only used for regression tasks and determines the shape of the input membership functions, that are generated from the cluster's cuboids. A value of 2 leads to a gaussian shape and should be a reasonable choice. Moreover, the resulting accuracy was worse for a value of 1 on the benchmark Temp.
- As described in section 2.3.3, the parameter $N_{G\,Max}$ is used as a threshold for the number of data tuples with the same output value, that are processed simultaneously. It reduces the required runtime for large sample sizes. This parameter was initially set to a value of 250. In the experiments ExpC in section 4.5 this choice is supplementary questioned and the impact of this parameter on the learning process is examined.
- The parameter N_{Bins} determines the number of intervals used to equidistantly or equifrequently discretize continuous inputs for feature weight calculation. The values usually used vary from just a few up to 20 or more intervals. Here we have chosen to use a value of 10 as it is done in [WD95, DKS95].
- The parameter δ_{Symb} is only needed, as described in section 1.8.1, together with the range or 4σ-normalization. It determines the normalization factor δ_j, that is used to down weight the influence of nominal inputs. The value was chosen according to the results in [WM97a], where a value of 0.3^{-1} performed best in combination with an IBL algorithm. Note that this parameter is not used for the d_{avr}-normalization, as all normalization factors are calculated from the average component wise distances there. Remarkably the resulting average normalization factors of the nominal inputs for the benchmarks Australian ($\overline{\delta} = 2.8$) and Hepatitis ($\overline{\delta} = 2.5$) are approximately the same.
- The parameters $W_{Kernel\,Min}$ and W_{Kernel} are only used for classification tasks and control the neighborhood area for the prediction as described in section 2.3.5. In favor of a small number of free parameters it does not seem advisable to use two parameters to control one strategy. Thus the value of $W_{Kernel\,Min}$ was chosen as 0.

4.3 Experiments ExpA: Design Parameters

4.3.1 Experiment Description

Table 4.3 lists the free parameters of the PNC 2 that were classified as design parameters, i.e. the parameters for which a fixed setting is chosen based upon experiments with some selected benchmarks.

Table 4.3: Overview of design parameters and possible settings.

Design parameter	**Alternatives**	**Reference**
Recalculation cluster output values	Off, On*	2.3.5
Merge test	Single, Doubled*	2.2.2
Normalization method	Range*, 4σ, d_{avr}	1.8.1
Hitrate to rate rules	Off MFC*, Off MTB, On, On LC	1.2
Discretization method	Equidistant, Equifrequent*	1.8.2
Feature weights	On*, Off	1.8.2
CSFS	On*, Off	2.3.4

Note: * = default alternative, MTB = mass tie breaking, MFC = most frequent class, LC = laplace corrected

In fact one should consider all possible combinations of the various alternatives. However this would require unreasonable much work. Thus a default alternative is set for each possible design parameter and the prediction accuracy achieved with this default alternative is compared with all those accuracies, that result, if exactly one parameter is not set to the default value.

Three strategies were employed to decide upon which alternative is used as default. Firstly, it was tried to identify an alternative, that should reasonably be better than the other ones. Second the alternative most usually used and third the alternative that should lead to the smaller models was preferred.

All remaining free parameters of the PNC 2 cluster algorithm are strategy parameters. Here they are set to the fixed values as stated in table 4.4. These values were chosen with respect to some quick experiments with the development benchmarks. Note that this predetermination of the strategy parameters without prior examinations may be problematically. It would be better to already chose default parameter sets for the strategy parameters here and to use the parameter tuning to select the set that performs best in combination with the actual design parameters. However this would requires much more work and computational power.

Table 4.4: Values of the strategy parameters used for the experiments EXPA.

Benchmark	N_{Int}	η	w_{COD}	p_{min}	W_{Kernel}	σ_{MSF}
TEMP	15	0.5	0	2	–	0.001
All other	–	0.5	3	2	2	–

The prediction accuracy is determined using a N_R-fold repetition with approximately equally sized learn and test data samples. The MCE criterion is used for classification and the MAE criterion is used for regression tasks. For each variant the probability, that the observed difference to the default variant is accidentally, is calculated using the two sided t-test for the case of paired samples (Appendix C). An overview of the benchmarks is given in table 4.5. Note that only 1000 out of 8124 data tuples are used for the benchmark MUSHROOMS.

Table 4.5: Overview for the experiments EXPA.

Benchmark	N	m	nom	cont	N_R
AUSTRALIAN	690	14	8	6	100
HEPATITIS	155	19	13	6	100
MUSHROOMS	1000	22	22	0	100
SEG	2130	19	0	19	25
TEMP	1126	3	0	3	25

Note: *nom* and *cont* denote the number of nominal resp. continuous inputs
Note: N is the total sample size, i.e. $N_L + N_T$

4.3.2 Evaluation

The detailed results are documented in appendix B. Relative differences $\triangle Q_T[\%]$ to the default variant are considered to be significant if the error probability P_{H0} is less than 10%. An error probability of less than 0.1% is stated as 0%. The results for the various design parameters are discussed in the following itemization. Thereby the terms *better* and *worse* are used to indicates that a particular alternative for a design parameter leads to a better resp. worse prediction accuracy when compared with the default alternative. The design parameters finally chosen are listed in table 4.6.

Table 4.6: Summary of design parameter values finally chosen.

Design Parameter	**Alternative**
Recalculation cluster output values	On
Merge test	Doubled
Normalization method	Range
Hitrate to evaluate rules	Off MFC
Discretization method	Equifrequent
Feature weights	On
CSFS	*Strategy parameter (decision deferred)*

- *No recalculation of cluster output values.*

Benchmark	$\triangle Q_T[\%]$	P_{H0}
TEMP	−51.7%	0.0%

This design parameter applies only for regression tasks and thus can only be made based upon the results for the benchmark TEMP. The prediction accuracy is about 50% worse if the cluster output values are not recalculated. Thus recalculation is chosen.

- *Normalization.*
  - *4σ-normalization.*

Benchmark	$\triangle Q_T[\%]$	P_{H0}
HEPATITIS	−3.1%	4.5%
SEG	−4.2%	4.1%
TEMP	1.0%	0.2%

  - *d_{avr}-normalization.*

Benchmark	$\triangle Q_T[\%]$	P_{H0}
AUSTRALIAN	−0.2%	8.5%
HEPATITIS	−4.1%	2.2%
MUSHROOMS	−9.6%	0.8%
SEG	−5.0%	5.0%
TEMP	3.5%	0.2%

The 4σ- and d_{avr}-normalization are usually significantly worse than the range normalization. Surprisingly clear is the result for the d_{avr}-normalization. In [HC99], on the contradiction, it is reported that the d_{avr}-normalization can lead to better prediction accuracy if used together with a kNN algorithm. However the results here are not directly comparable as a normalization factor was used to down weight the influence of nominal inputs. Thus the experiments with the range- and 4σ-normalization were repeated with $\delta_{Symb} = 1$ for the benchmarks AUSTRALIAN and HEPATITIS. This setting leads for the benchmark HEPATITIS to a prediction accuracy which is 5.1% ($P_{H0} = 0.6\%$) worse than before. However, the range normalization is selected, as this alternative is clearly better than the others if the normalization factor for nominal inputs is chosen to $\delta_{Symb} = 0.3^{-1}$.

- *Single merge test.*

Benchmark	$\triangle Q_T[\%]$	P_{H0}
AUSTRALIAN	0.5%	0.0%
HEPATITIS	−10.5%	0.0%
MUSHROOMS	−470.7%	0.0%
SEG	−13.0%	0.0%
TEMP	−35.3%	0.2%

The single merge test leads, as expected, to a bigger merge rate and thus produces smaller models. The average hitrate of the clusters is smaller, the degree of coverage, i.e. the proportion of the test data tuples, whose input vector is covered by at least one cluster, is higher. However, the doubled merge test is clearly superior and is chosen.

- *Hitrate for rule evaluation.*
  - *Off mass tie breaking (MTB).*

Benchmark	$\triangle Q_T[\%]$	P_{H0}
MUSHROOMS	−39.1%	0.0%

Ties are broken in favor of the cluster with the biggest mass if the MTB policy is used. For the benchmark MUSHROOMS this leads to a prediction accuracy, that is approximately 40% worse. This benchmark has only two output classes, whose prior probabilities are approximately 20% and 80%. The default alternative *Off MFC*, to predict the most frequent output class in case of a tie, seems to be the more robust tie breaking policy. More over the MTB policy requires, that the mass of each cluster is additionally stored, which causes bigger model sizes and less interpretability.

  - *Laplace corrected hitrate.*

Benchmark	$\triangle Q_T[\%]$	P_{H0}
HEPATITIS	0.5%	3.5%
MUSHROOMS	−38.5%	0.0%
SEG	−1.7%	9.1%
TEMP	−2.0%	0.0%

  - *Hitrate.*

Benchmark	$\triangle Q_T[\%]$	P_{H0}
SEG	1.6%	1.6%

Based on these results no rule rating is used for the PNC 2 cluster algorithm, as this had a negative impact on the prediction accuracy in most of the above experiments. At all events the ordinary hitrate comes in question, as this increases prediction accuracy significantly for one benchmark. However the rule's ratings would have to be additionally stored and thus the model size would increase.

- *Equidistant discretization.*

Benchmark	$\triangle Q_T[\%]$	P_{H0}
SEG	−8.1%	1.0%

The discretization method has, for most of the benchmarks, only a smaller impact on the resulting prediction accuracy, as it only slightly modifies the way how the feature weights are calculated. The equifrequent discretization is significantly better for the benchmark SEG and moreover it should be the more robust strategy. Thus it is chosen.

- *No feature weights.*

Benchmark	$\triangle Q_T[\%]$	P_{H0}
AUSTRALIAN	−2.8%	0.3%
HEPATITIS	−2.8%	9.1%
MUSHROOMS	−11.9%	0.8%

Feature weights, that are calculated similar to eqn. (1.35), are used together with the NGE and kNN algorithms in [WD95]. Similar results are reported there: The usage of feature weights, that were calculated based on the transinformation, can have a significant positive effect on the resulting prediction accuracies for some benchmarks and do not make things worse for the other ones. Thus feature weights are always used for the PNC 2 cluster algorithm.

- *Context sensitive feature selection (CSFS).*

Benchmark	$\triangle Q_T$[%]	P_{H0}
AUSTRALIAN	0.5%	0.1%
HEPATITIS	2.8%	4.6%

The prediction accuracy is slightly better for all benchmarks, but this difference is only significant for the benchmarks AUSTRALIAN and HEPATITIS. On the other side the application of the CSFS leads to models, that are smaller by a factor of approximately 3.6. Thus the decision is deferred here and the usage of the CSFS is from now on considered as a strategy parameter, that is tuned for the particular benchmark at hand.

4.4 Experiments ExpB: Strategy Parameters

4.4.1 Experiment Description

For each strategy parameter some default values are defined and the default parameter sets are generated from these values by building all possible combinations. The default values for all parameters – except ω_{COD} – are defined by choosing some reasonable values for each parameter as stated in table 4.7.

Table 4.7: Default parameter sets.

N_{Int}	η	ω_{COD}	p_{min}	W_{Kernel}	CSFS	σ_{MSF}
{10, 15, 20}	{0, 0.5, 1}	*see below*	{7, 4, 2}	{1, 2, 3}	{0, 1}	{0.01, 0.001, 0.0001}

For the parameter ω_{COD} an interval of possible values is defined depending on the number of inputs m as

$$[0, \omega_{max}] \quad \text{with} \quad \omega_{max} \approx \frac{m}{3}. \tag{4.1}$$

This interval may be further modified by avoiding the smaller values for benchmarks with a lot of inputs and by avoiding the higher values for benchmarks with a lot of nominal inputs. Three to five values are then chosen within this interval as stated in table 4.8.

The prediction accuracy is estimated for each parameter set using a N_R-fold repetition with approximately equally sized learn and test data samples. The MCE criterion is used for classification and the MAE criterion is used for regression tasks. Table 4.8 gives an overview of the sample sizes N, of the number of repetitions N_R done and of the number and type of inputs. Note that only 1000 out of 8124 data tuples are used for the benchmark MUSHROOMS.

Table 4.8: Overview for the experiments ExpB.

Benchmark	N	m	nom	cont	N_R	ω_{COD}
AUSTRALIAN	690	14	8	6	400	{0, 1.5, 3, 4.5}
HEPATITIS	155	19	13	6	100	{0, 2, 4, 6}
MUSHROOMS	1000	22	22	0	1500	{0, 1.5, 3, 4.5}
SEG	2130	19	0	19	150	{1, 2.5, 4, 5.5}
TEMP	1126	3	0	3	100	{0, 0.5, 1}

Note: *nom* and *cont* denote the number of nominal resp. continuous inputs
Note: N is the total sample size, i.e. $N_L + N_T$

The parameter sets are sorted with respect to the corresponding prediction accuracy. The value of N_R is chosen such that the best parameter set is the same if only $0.5\,N_R$ repetitions would have been done. This procedure aims to reduce the following over fitting problem: The prediction accuracies $\widehat{Q}$ are only estimates of the true values Q^* and thus they will scatter with an unknown distribution around the real values. The sorting introduces a kind of bias, as the best parameter set will more likely correspond to an overly optimistically estimate than vice versa.

4.4.2 Evaluation

Table 4.9 lists the two best parameter sets with and without application of the context sensitive feature selection. Additionally the worst parameter set is stated. Thereby $\widehat{Q}_T$ denotes the prediction accuracy, K resp. K_{Red} denotes the number of generated clusters without resp. with reduction of those clusters whose mass does not exceed the threshold p_{min} and $\varnothing\ m$ is the average number of inputs in the rules' premise.

Table 4.9: Results of the experiments ExpB.

Benchmark	N_{Int}	η	w_{COD}	p_{min}	W_{Kernel}	σ_{MSF}	CSFS	$\widehat{Q}_T$	K	K_{Red}	$\varnothing\ m$
Australian	–	1	4.5	7	1	–	1	14.53 %	168.3	3.8	5.2
	–	1	4.5	7	2	–	0	14.59 %	168.3	3.8	14.0
	–	0	4.5	7	1	–	1	33.40 %	264.0	1.9	5.8
Hepatitis	–	0	4	2	3	–	1	17.88 %	19.8	9.3	4.9
	–	0.5	0	7	3	–	0	19.62 %	5.6	2.1	19.0
	–	0	6	7	1	–	1	29.58 %	32.7	1.4	5.2
Mushrooms	–	0	1.5	2	1	–	0	0.36 %	5.7	5.3	22.0
	–	0	3	2	1	–	1	0.47 %	8.1	6.9	4.6
	–	1	0	7	3	–	1	5.91 %	2.9	2.7	3.9
Seg	–	0	1	2	3	–	0	4.19 %	32.3	26.3	19.0
	–	0	1	2	3	–	1	4.33 %	32.3	26.3	2.5
	–	0	5.5	7	1	–	0	11.73 %	147.6	30.7	19.0
Temp	10	0	0.5	2	–	0.0001	1	1.159	44.2	29.5	1.4
	10	1	0.5	2	–	0.001	0	1.269	28.1	19.4	3.0
	20	0	0.5	7	–	0.001	1	2.903	72.4	9.1	1.5

The following conclusions are drawn:

- η. A choice of $\eta \to 1$ leads to a higher merge rate and thus to a smaller number of generated clusters K. The mass of the clusters will tend to be bigger, whereby the reduction effect resulting from the parameter p_{min} will be less.
- w_{COD}. The choice of $w_{COD} \to \omega_{max}$ leads to a smaller merge rate. A value much to large causes a very low merge rate and thus the number of generated clusters will almost be the same as the number of learn data tuples. For the benchmark Australian this leads to instability in the sense, that small differences of one parameter value can cause a significantly worse prediction accuracy: The best and the worst parameter set differ only in the value of the parameter η. Thus a minimal merge rate should be considered as additional condition for the parameter tuning. This also should decrease the necessary runtime of the Pnc 2 cluster algorithm, as a higher merge rate implies, that less merge tests need to be done. Altogether the designated range of values seems to be sufficient, as the values, which occur in the best parameter sets, are mostly within the range of values.
- p_{min}. A choice of $p_{min} \to 7$ causes a higher reduction of clusters and thus leads to smaller models. This effect can be observed best in combination with bigger values of the parameter w_{COD}, as then, due to the smaller merge rate, there will be many cluster with a small mass.
- W_{Kernel}. As expected, a choice of $W_{Kernel} > 1$ can have a positive effect on prediction accuracy. However sometimes a setting of $W_{Kernel} = 1$ yields the best results. Therewith this parameter is similar to the parameter k of the kNN algorithm, that should also be set to $k = 1$ for some benchmarks to get the best results.
- *Context sensitive feature selection (CSFS).* Partly the application of the CSFS leads to better prediction accuracies. This proves, that the previous decision to automatically tune this parameter was fortunate.

4.5 Experiments ExpC: Runtime and the Parameter $N_{G\,Max}$

4.5.1 Experiment Description

Firstly, within the scope of the experiments ExpC1, the effect of the parameter $N_{G\,Max}$ on runtime, prediction accuracy and model size is examined. The parameter $N_{G\,Max}$ is described in section 2.3.3. It is a threshold for the number of learn data tuples with the same output value that are processed simultaneously. A fixed value of 250 was chosen as a design decision in section 4.2. Secondly, within the scope of the experiments ExpC2, the effect of the learn data sample size on runtime, prediction accuracy and model size is examined. Different from the other preliminary studies, now the benchmarks Dna, Letter and Wave are used, as larger sample sizes are needed. However the results are not used in the benchmark comparison study in section 4.7 as this would violate the conditions of a tough validation. The strategy parameters are set to the fixed values as stated in table 4.10. The experiments were done using a Pentium iii 700 MHz (Letter and Wave) resp. using an Amd Duron 1 GHz (Dna).

Table 4.10: Strategy parameter values for the experiments ExpC1 and ExpC2.

η	ω_{COD}	p_{min}	W_{Kernel}	CSFS
0.5	3	1	2	0 *or** 1

* The setting that leads to the best prediction accuracy is chosen for each benchmark.

For the experiments ExpC1 the value of $N_{G\,Max}$ is set to the values 50, 75, 100, 125, 250, 500 and 1000 one after the other. Accuracy is estimated using a N_R-fold repetition with the learn and test data sample sizes as stated in table 4.11. Note that also the maximal number of expectable learn data tuples with the same output value $\widetilde{N}_{G\,Max}$ is stated. Increasing the value of $N_{G\,Max}$ above $\widetilde{N}_{G\,Max}$ should not have any effect.

Table 4.11: Overview of experiments ExpC1.

Benchmark	N	m	nom	cont	N_L	N_T	N_R	$\widetilde{N}_{G\,Max}$
Dna	3175	60	60	0	1700	1475	200	850
Letter	20000	16	0	16	16000	4000	75	650
Wave	5000	21	0	21	2550	2450	50	850

Note: *nom* and *cont* denote the number of nominal resp. continuous inputs
Note: N is the total sample size, i.e. $N_L + N_T$

For the experiments ExpC2 the fixed settings $N_{G\,Max} = 125$ and $N_{G\,Max} = \infty$ are used. Again a N_R-fold repetition is done to estimate the accuracy for the various number of learn data sample sizes. Note, that there are less test data tuples, if the learn data sample size increases and thus the estimated accuracy will become less accurate. The number of repetitions was chosen as stated in table 4.12 for the parameter setting $N_{G\,Max} = 125$. For the setting $N_{G\,Max} = \infty$ only two repetitions were done due to increasing computational costs.

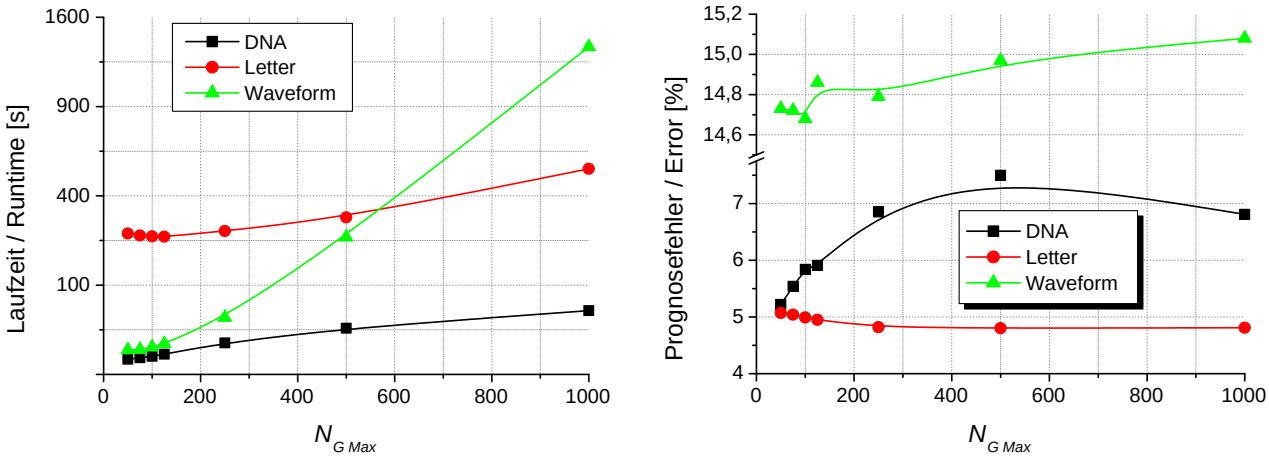
Table 4.12: Overview of experiments ExpC2.

Benchmark	N	m	nom	cont	N_R	$N_{L\ 100\%}$
Dna	3175	60	60	0	40	3060
Letter	20000	16	0	16	10	18000
Wave	5000	21	0	21	10	4590

Note: *nom* and *cont* denote the number of nominal resp. continuous inputs
Note: N is the total sample size, i.e. $N_L + N_T$

4.5.2 Evaluation

Runtime, model size and accuracy are shown in the figures 4.1 and 4.2 for the different values of the parameters $N_{G\,Max}$.



(a) **(b)**

Figure 4.1: Runtime (quadratic scale)(a) and accuracy $\widehat{Q}_T$ (b) of a model learned with the PNC 2 cluster algorithm for different values of the parameter $N_{G\,Max}$.

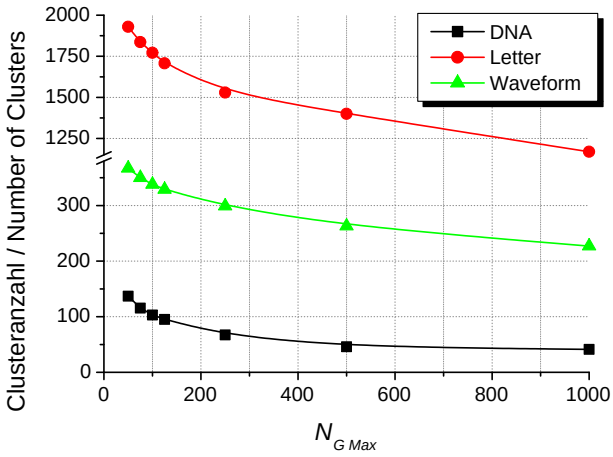


Figure 4.2: Number of clusters of a model learned with the PNC 2 algorithm for different values of the parameter $N_{G\,Max}$.

Runtime increases and model size decreases with increasing values of the parameter $N_{G\,Max}$. Surprisingly however the prediction accuracy increases only for the benchmark LETTER, whereas it gets worse for the benchmarks DNA and WAVE. This indicates, that the usage of the parameter $N_{G\,Max}$ introduces a kind of bagging situation, which can lead – as discussed in the following paragraph – to higher prediction accuracies. The choice $N_{G\,Max} = 125$ seems to be a reasonable trade-off between prediction accuracy on the one hand and required runtime and model size on the other hand. However, the previously chosen value of 250 is still used in the following benchmark comparison study in order to fulfill the conditions of a tough validation.

Runtime, accuracy and model size is shown in the figures 4.3 and 4.4 for various learn data sample sizes. Thereby 100% of the learn data tuples correspond to the value of $N_{L\ 100\%}$ as stated in table 4.12. It can be seen, that runtime increases approximately quadratically with the learn data sample size.

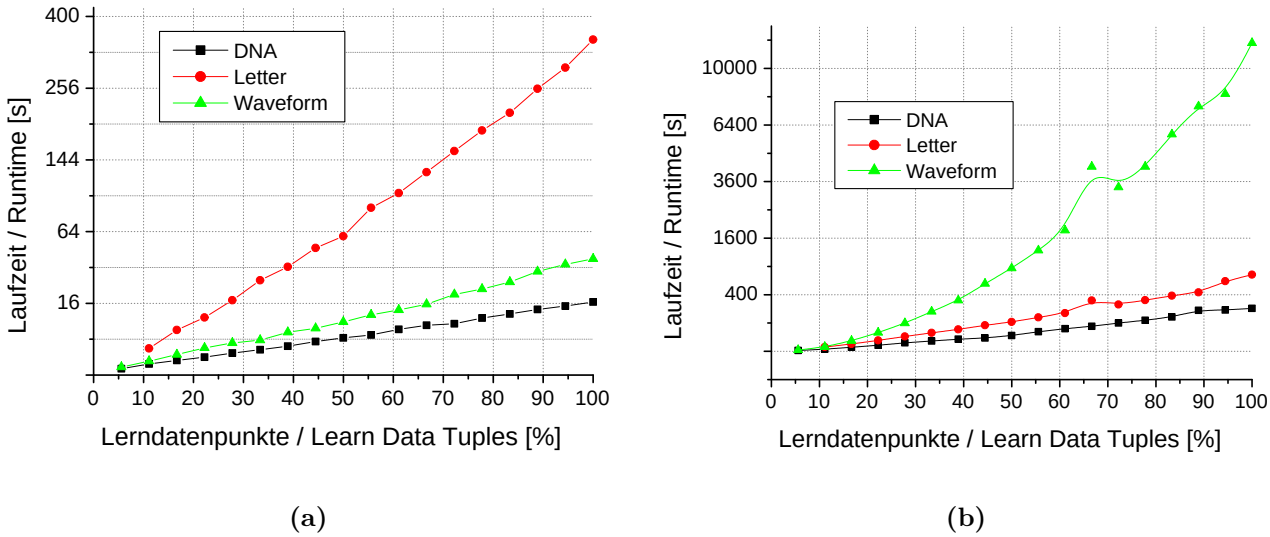


Figure 4.3: Runtime (quadratic scale) to learn a model with the PNC 2 cluster algorithm with respect to the learn data sample size for the parameter setting $N_{G\,Max} = 125$ (a) and $N_{G\,Max} = \infty$ (b).

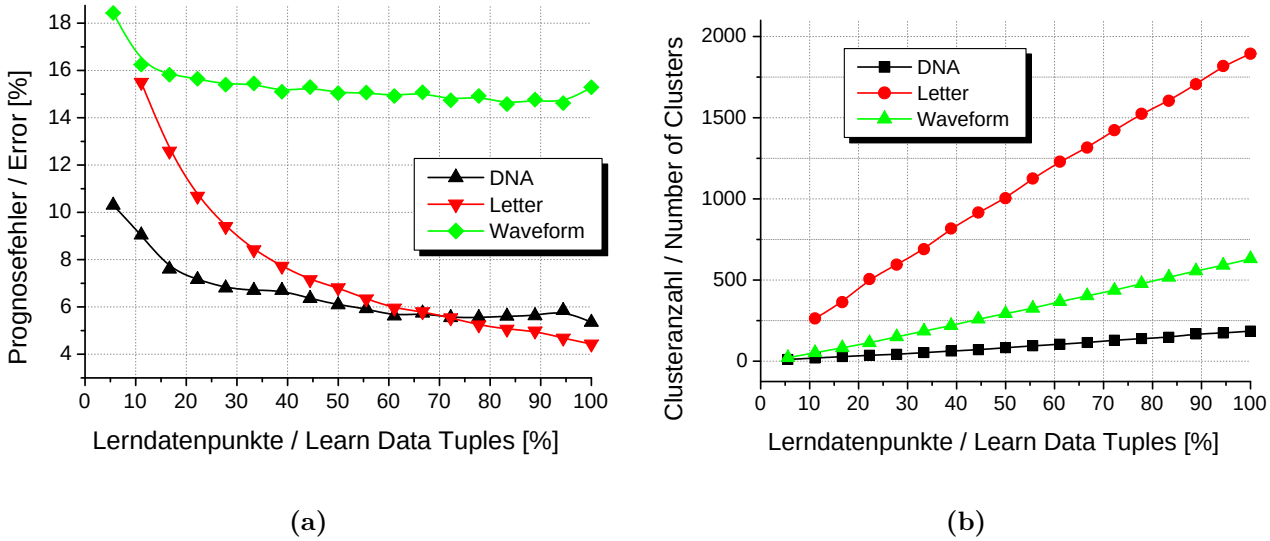


Figure 4.4: Accuracy $\widehat{Q}_T$ and number of clusters for a model learned with the PNC 2 cluster algorithm with respect to the learn data sample size for a parameter setting $N_{G\,Max} = 125$.

4.5.3 Discussion

Recently increasing work was spent to discuss the question if and how the combination of several prediction models, a so called *ensemble of classifiers*, can be used to improve prediction accuracy [KHDM98, Die98, BK99]. Therefore several models are learned based upon the same learn data sample and the predictions of the individual models are usually combined using a weighted or unweighted voting. This leads to an improved prediction accuracy if the errors of the individual models are uncorrelated among each other and if the individual error rates are below 50% [Die98]. To learn models, whose prediction errors are uncorrelated, it is firstly necessary, that the predictions of the individual models differ at least in some parts of the input space. Thus the models are learned using different learn data samples, using different learning algorithms or just using different parameterizations for one and the same learning algorithm[1].

The usage of the parameter $N_{G\,Max}$ has obviously the same effect as if several different models on different

[1]Several so called *bagging* and *boosting* methods used to divide a learn data sample into several smaller sub-samples are described in [BK99]. The various methods are compared experimentally with each other with respect to several different learning algorithms.

learn data samples were learned. This may explain the positive effect, that a a smaller value for $N_{G\,Max}$ has on the prediction accuracy.

4.6 Experiments ExpE: Tuning Parameters

The tuning parameters define the exact proceeding for the automated parameter tuning. They are defined based upon the results from the following experiments. Accuracy is estimated using a N_R-fold repetition with approximately equally sized learn and test data samples. The number of repetitions N_R is stated in table 4.13. The MCE criterion is used for classification and the MAE criterion is used for regression tasks. The default parameter sets for the strategy parameters are set according to the tables 4.7 and 4.8. Four individual experiments are done. These are described and evaluated in the following itemization. For a lack of space the results are only summarized and not documented in detail. The tuning parameters finally chosen are listed in table 4.15.

Table 4.13: Choice of N_R for the experiments ExpE.

Australian	Hepatitis	Mushrooms	Seg	Temp
100	200	50	100	100

4.6.1 Experiments ExpE1 – Accuracy Estimation Method

Firstly the learn sample accuracy is used as objective for the parameter tuning. Three different estimation methods for this accuracy are considered here: The 10-fold cross-validation and the 10 resp. 50-fold repetition. For the latter approach the learn data sample is split at the rate of 80 to 20.

Evaluation: There are no differences in the resulting prediction accuracies except for the benchmark Seg, where the 50-fold repetition leads to a prediction accuracy of 4.3%, whereas the prediction accuracy when using a 10-fold cross-validation is only 4.64%. This difference is significant – if evaluated with the t-test for the case of paired samples as described in appendix C – with an error probability of 0.1%. More over the model size K_{Red} is approximately 25% smaller.

Probably a 10-fold cross-validation or repetition is completely sufficient for the benchmarks with the smaller sample sizes. A 50-fold repetition does not achieve any improvements there – a more precise estimation of the learn sample accuracy is affected by over fitting effects. However the 50-fold repetition can be better if the sample size is bigger. Thus the 50-fold repetition is used – as not stated otherwise – in all following experiments.

4.6.2 Experiments ExpE2 – Additional Constraints

The *minimal merge rate* λ_{min} and the *maximal model size* κ_{max} are introduced motivated by the observations made in connection with the experiments ExpB. There, for the benchmark Australian, an instability of a parameter set could arise if a high value of ω_{COD} was chosen. The merge rate is calculated as

$$\lambda = \frac{K}{N_L} \ , \tag{4.2}$$

whereas K denotes the number of clusters before any reduction and N_L is the learn data sample size. The model size is calculated as

$$\kappa = K_{Red}\ \emptyset m \ , \tag{4.3}$$

with K_{Red} being the number of clusters after reduction and $\emptyset m$ being the average number of inputs in the rules' premises. If the model size is to big or if the merge rate is to low for any of the repetitions done, then the corresponding parameter set is discarded.

To additionally save runtime simulations are skipped, if the corresponding parameter set will probably be ignored due to the minimal merge rate or maximal model size. If a parameter set a is discarded, then it is assumed, that a parameter set b for which

$$(\eta_a \leq \eta_b) \ \wedge \ (\omega_{COD\,a} \geq \omega_{COD\,b}) \ \wedge \ (p_{min\,a} \leq p_{min\,b}) \ \wedge \ (CSFS_a \leq CSFS_b) \tag{4.4}$$

yields, will also be discarded[2]. The simulation for the different parameter sets are ordered such that the parameter sets, that probably will produce smaller models, are simulated first.

Some settings for λ_{min} and κ_{max} are experimentally compared for the benchmark AUSTRALIAN. The results are listed in table 4.14, whereas T_{Tune} denotes the time required for the whole parameter tuning on a PENTIUM III 700 MHz system. The choice of $\lambda_{min} = 50\%$ and $\kappa_{max} = 10\%$ leads to significantly smaller models and saves approximately 80% runtime. Therewith this seems to be a good setting, which is used in all following experiments.

Table 4.14: Results of the experiments EXPE2.

λ_{min}	κ_{max}	$\widehat{Q}_T$	K	K_{Red}	ø m	T_{Tune}
100 %	100 %	15.01 %	95.2	9.2	10.7	14m35s
100 %	10 %	14.90 %	88.2	5.8	8.5	13m33s
50 %	10 %	14.94 %	53.0	5.7	9.1	3m21s

4.6.3 Experiments ExpE3 – Tuning Objective

Up to now the learn sample accuracy was used as tuning objective. Tentatively the *learn sample rank*, that is estimated as described in section 3.2.1, is used instead. However, there are no significant differences of the resulting prediction accuracy and thus the original approach is kept.

4.6.4 Experiments ExpE4 – Context Sensitive Feature Selection

One of the tuned strategy parameters is the activation of the context sensitive feature selection (CSFS). It is activated for the benchmark TEMP in more than 95% of all simulations, but for the other benchmarks is is activated far less. Now the CSFS is tentatively permanently activated. On the one hand this leads to prediction accuracies that are about 5% worse, but on the other hand the models are about 30% smaller. Considering the worse prediction accuracy the original approach is kept.

Table 4.15: Tuning parameters finally chosen.

Objective	learn sample accuracy MCE/MAE
Estimation learn sample accuracy	50-fold repetition*
$N_{L\,Tune}$ resp. $N_{T\,Tune}$	0.8 N_L resp. 0.2 N_L *
Min. merge rate λ_{min}	50 %
Max. model size κ_{max}	10 %

* A smaller number of repetition resp. tuning learn and test sample sizes is used for bigger learn sample sizes to reduce the computational costs of the parameter tuning.

4.7 Benchmark Studies

The PNC 2 cluster algorithm is compared with the most similar existing approaches, namely with the NGE, the RISE and the kNN algorithm, in an extensive benchmark study. For further details on the particular algorithms see the sections 2.4 and 1.3.

The results for the PNC 2 cluster algorithm are obtained executing a tough validation as described in section 3.2.1, whereas the results for the other algorithms are taken from literature. Note, that this is only a suboptimal approach. It would have been better to make own tests with all algorithms under consideration, as this allows to compare the resulting accuracies with a t-test for the case of paired samples. However, this would have been much more work.

[2]Note, that this must not generally be the case. A larger value of the parameter ω_{COD} may lead to a lower merge rate, thus more small clusters will exists and the reduction by the threshold p_{min} may increase. Thus a larger value of the parameter ω_{COD} may actually lead to a smaller model size. However, this effect is not wanted anyway.

The prediction accuracy of the PNC 2 cluster algorithm is estimated using a N_R-fold repetition. The number of learn and test data tuples used to compare with the various algorithms and the standard values used for the strategy parameter ω_{COD} are stated in table 4.16. The design decisions and design parameters are set according to the tables 4.2 and 4.6. The standard values for the strategy parameters are chosen according to table 4.7 and the tuning parameters are set according to table 4.15. Relatively big learn data samples are used for the benchmarks DNA, LETTER and MUSHROOMS. Thus, to save computational costs, different from table 4.15, the particular learn data sample is split in approximately equally sized tuning learn and test data samples[3] and the number of repetitions to estimate the learn sample accuracy is reduced to 10.

Right now the PNC 2 cluster algorithm has no special policy of how to treat missing feature values. Thus, as a simple work around, missing feature values are replaced in a pre-processing step by an additional symbol for nominal inputs and by a value, that is outside of the particular input's range, for continuous inputs. This simple solution may be unfavorable for continuous inputs, as the component wise distance of a missing feature value to the different possible known values will be different.

Table 4.16: Standard values for the strategy parameter ω_{COD} and learn and test data sample sizes for the comparison of the PNC 2 with the NGE, the RISE and the kNN algorithm.

		NGE and kNN		RISE	
Benchmark	ω_{COD}	N_L	N_T	N_L	N_T
CHESS	{0, 2, 4, 6}	–	–	2109	1087
DNA	{2, 4, 6, 8}	–	–	2117	1058
CLEVELAND	{1, 2, 3, 4, 5}	212	91	202	101
HEPATITIS	{0, 2, 4, 6}	–	–	103	52
IRIS	{0, 0.5, 1, 1.5}	105	45	100	50
LETTER	{1, 2, 3, 4, 5}	16000	4000		
MUSHROOMS	{0, 1.5, 3, 4.5}	–	–	5333	2167
VOTES	{1, 2, 3, 4}	305	130	290	145
WAVE	{1, 2.5, 4, 5.5, 7}	300	4700	–	–
WAVE+NOISE	{4, 6, 8, 10, 12}	300	4700	–	–
WINE	{1, 2, 3, 4, 5}	–	–	118	60

4.7.1 Comparison with the NGE and the kNN Algorithm

Experiment Description The NGE algorithm and several variants are compared experimentally with several variants of the kNN algorithm in [WD95]. Based upon this study the predecessor of the PNC 2 cluster algorithm, the PNC algorithm, was compared in [Hae01b] and [Hae01a] with the NGE and the kNN algorithm. Always the variant, that reached the best prediction accuracy on a particular benchmark, was taken and compared to the PNC algorithm. This was a little bit unfair to the PNC algorithm, as it cannot be guaranteed, that it would have been possible, to select the very variant, that reaches the best prediction accuracy on test data, using an automated parameter tuning within the learn data.

The different nearest-neighbor approaches used in [WD95] are listed in table 4.17. Two variants use, as the PNC 2 cluster algorithm, feature weights, that are calculated based upon the mutual information. From these two, the variant $k\text{NN}_{CV\,MI}$ follows the very approach to set the parameter k using a parameter tuning within the learn data. Insofar, the variant NN_{MI} is included as a special case with a setting of $k = 1$ in the variant $k\text{NN}_{CV\,MI}$. Thus now, for the comparison of the PNC 2 with the kNN algorithm, only the variant $k\text{NN}_{CV\,MI}$ needs to be considered.

For the comparison with the NGE algorithm it is not possible to select a single variant. Thus further on the very variant is chosen, that reaches the best prediction accuracy for the particular benchmark. The resulting prediction accuracies for the variant OBNGE are additionally stated, as this variant is quite similar to the basic approach of the PNC 2 cluster algorithm.

The results in [WD95] are obtained using a 25-fold repetition with the learn and test data sample sizes chosen as stated in table 4.16. The standard error of the prediction accuracy stated in [WD95] is converted back, according to eqn. (3.5), into the standard deviation of the prediction accuracy. Note, that there may be some inaccuracies as the values in [WD95] are only given with one decimal place.

[3] Further more, for the benchmark LETTER, only 8000 of the 16000 data tuples of the learn data sample are used.

Table 4.17: The four kNN-variants, that were used in [WD95].

$k\text{NN}_{CV}$	kNN algorithm; the parameter k is chosen using a 10-fold cross-validation within the learn data sample
NN	kNN algorithm with a fixed value of $k = 1$
$k\text{NN}_{CV\,MI}$	The same as $k\text{NN}_{CV}$, but feature weights, calculated based upon the mutual information, are used
NN_{MI}	The same as NN, again using feature weights as $k\text{NN}_{CV\,MI}$

The results are given in table 4.18. Thereby K denotes the number of clusters before and K_{Red} the number of clusters after reducing those clusters, whose mass does not exceed the threshold p_{min} and $\varnothing\ m$ is the average number of inputs used per cluster. As prediction accuracy the mean classification error MCE with respect to the test data samples and its deviation is given.

Table 4.18: Comparison of the PNC 2 with the NGE and the kNN algorithm.

	$k\text{NN}_{CV\,MI}$	NGE_{Best}		OBNGE	
Benchmark	$\widehat{Q}_T$	$\widehat{Q}_T$	K	$\widehat{Q}_T$	K
CLEVELAND	18.3 % ± 3.0 %	21.5 %	57	28.7 %	63
IRIS	4.9 % ± 2.5 %	5.3 %	6	6.8 %	4
LETTER	3.4 % ± 0.– %	8.7 %	2560	–*	–*
VOTES	4.6 % ± 2.0 %	5.3 %	46	11.2 %	38
WAVE	17.4 % ± 4.5 %	25.9 %	116	29.7 %	9
WAVE+NOISE	17.6 % ± 3.0 %	29.9 %	20	35.5 %	5

* not specified as, according to [WD95], a simulation was not possible due to the high computational costs

	PNC 2				
Benchmark	$\widehat{Q}_T$	K	K_{Red}	$\varnothing\ m$	N_R
CLEVELAND	18.12 % ± 3.6 %	74.9	6.3	7.0	250
IRIS	4.54 % ± 2.6 %	9.3	3.9	1.5	250
LETTER	5.71 % ± 0.4 %	1566	1109	7.6	25
VOTES	4.93 % ± 1.5 %	34.4	4.5	4.4	150
WAVE	18.10 % ± 1.1 %	41.7	22.0	10.1	150
WAVE+NOISE	17.21 % ± 0.9 %	37.1	27.6	16.1	150

Evaluation The PNC 2 cluster algorithm always reaches superior prediction accuracies if compared with the very NGE variant, that performs best for the particular benchmark. The variant OBNGE has very poor prediction accuracies. This is remarkable, as the OBNGE and the PNC 2 employ the same basic approach. However, the OBNGE algorithm does not have any mechanisms to circumvent the COD problem. Due to the poor results of the OBNGE, this approach was discarded in [WD95]. Now the results obtained here show, that this approach is quite good. It is only necessary to combine it with an effective anti COD mechanism and some additional strategies.

The prediction accuracy of the $k\text{NN}_{CV\,MI}$ algorithm is slightly excelled for some of the benchmarks – for some other benchmarks it is slightly worse. A big advantage of the PNC 2 cluster algorithm is the size of the generated models, that are always much smaller. Note, that the $k\text{NN}_{CV\,MI}$ needs to store all learn data tuples.

The prediction accuracy of the PNC 2 cluster algorithm for the benchmark WAVE+NOISE is approximately 5% better than for the benchmark WAVE. This difference is significant – if evaluated with a t-test for the case of paired samples – with a vanishing error probability. The two benchmarks are identical, except that the benchmark WAVE+NOISE contains 19 additional inputs, that contain nothing but random white noise. Thus one would have expected the opposite result. By further examinations, it was possible to find out, that the

additional irrelevant inputs cause an individual adjustment of the value of the parameter ω_{COD} with respect to the number of tuples merged in a cluster: If many tuples are merged in a cluster, then most of the irrelevant inputs' range will be covered and thus almost no data tuples will lie outside with respect to the irrelevant inputs. This effect should be examinated to further improve the PNC 2 cluster algorithm.

4.7.2 Comparison with the RISE Algorithm

Experiment Description In [Dom96] the RISE algorithm is experimentally compared with several other learning algorithms for various benchmarks. Thereby the prediction accuracies are obtained using a 50-fold repetition with the learn and test data sample sizes chosen as stated in table 4.16. From the benchmarks used in [Dom96] those, that were already used in this work, and three other randomly chosen benchmarks are taken.

N_{Mem} denotes the memory, that is needed to store the learned model. For the PNC 2 cluster algorithm it results from the number of clusters K_{Red} and from the average number of inputs used per cluster $\varnothing\ m$ as

$$N_{Mem} = K_{Red}(\varnothing m + 1)\ . \tag{4.5}$$

The results are given in table 4.19. Thereby K denotes the number of clusters before and K_{Red} the number of clusters after reducing those clusters, whose mass does not exceed the threshold p_{min} and $\varnothing\ m$ is the average number of inputs used per cluster. As prediction accuracy the mean classification error MCE with respect to the test data samples and its deviation is given.

Table 4.19: Comparison of the PNC 2 with the RISE algorithm

	RISE		PNC 2					
Benchmark	$\widehat{Q}_T$	N_{Mem}	$\widehat{Q}_T$	N_{Mem}	K	K_{Red}	$\varnothing\ m$	N_R
CHESS	1.8 % ± 0.2 %	3064	2.50 % ± 0.8 %	721	69.8	34.0	21.2	25
CLEVELAND	20.3 % ± 3.8 %	1461	17.97 % ± 3.5 %	46	71.5	5.8	6.9	250
DNA	6.9 % ± 1.6 %	8351	4.39 % ± 0.5 %	5117	285.4	208.0	23.6	25
HEPATITIS*+	21.7 % ± 5.7 %	1232	18.54 % ± 5.0 %	61	18.4	6.5	8.3	250
IRIS+	6.0 % ± 2.8 %	383	4.66 % ± 2.5 %	10	9.0	4.0	1.5	250
MUSHROOMS*	0.0 % ± 0.2 %	398	0.00 % ± 0.0 %	19	13.4	8.8	1.1	25
VOTES+	4.8 % ± 1.5 %	541	4.85 % ± 1.5 %	23	29.3	4.5	4.2	150
WINE+	3.1 % ± 2.0 %	896	3.36 % ± 2.0 %	52	8.8	6.4	7.1	250

* resp. + denotes: Benchmark was used to develop the PNC 2 resp. the RISE algorithm

Note: It is not clear, how the deviation stated for the benchmark MUSHROOMS for the RISE algorithm is possible.

Evaluation The PNC 2 cluster algorithm reaches better prediction accuracies than the RISE algorithm for most of the benchmarks. Moreover, the generated models are much smaller. However, a reduction mechanism for the RISE algorithm exists, that – with a slightly worse prediction accuracy – leads to a size reduction of approximately 90%. See section 2.4.2 for further details.

DOMINGOS states in [Dom96], that the prediction accuracy of the RISE algorithm cannot be worse than the one of the kNN algorithm. He argues, that the RISE algorithm starts with a state, that is identical to the kNN algorithm, as it is initialized by treating each learn data tuple as a trivial rule. Then rules are only generalized, if this does not have a negative impact on the prediction accuracy estimated using a leave-one-out cross-validation within the learn data sample. However, comparing the results for the benchmarks CLEVELAND, IRIS and VOTES given in the tables 4.18 and 4.19[4], one can see, that the RISE algorithm reaches worse prediction accuracies than the $k\text{NN}_{CV\,MI}$. Thus, to be concrete, the statement of DOMINGOS is only valid with respect to the kNN variant used within the RISE algorithm.

[4]Note, that not the same, but approximately the same learn data sample sizes were used to obtain the results stated in the tables 4.18 and 4.19.

Chapter 5

Summary and Outlook

5.1 Summary

Models for the input/output behavior of a system can be useful in various ways for simulation, object recognition and classification, prediction or simply to gather some knowledge about a given system. The data-based approach to build a model is quite fast and does not require much knowledge about the given system. First some data, that should reveal all relevant correlations or operating conditions, is collected from the given system. Then, based upon this data, a model is learned using a suitable learning algorithm. If, next to a high prediction accuracy, it is important, that the generated models are interpretable by a human, then it is advisable to use rule based models. These models consist of several IF-THEN rules, whereof each single rule describes a significant part of the system's input/output behavior.

Cluster algorithms are used to partition a given set of data tuples, with respect to a similarity measure, in different groups. These groups are called clusters. In the context of data-based rule induction it is possible to employ cluster algorithms for direct rule generation. Each cluster corresponds to a single rule. Most cluster algorithms work unsupervised, i.e. without any special treatment of an eventually existing output, and identify dense regions in the (input) space. In contrast, the aim of rule induction is to identify continuous regions of the input space where the corresponding output values are equal or at least similar. Thus the PNC 2 cluster algorithm is developed, that extends the hierarchical agglomerative cluster paradigm for the case of supervised rule induction. Next to a high prediction accuracy, special care was taken, that the resulting models can easily be interpreted by a human. Therefor only local decision strategies are used during the clustering and post-processing steps. This guarantees that each single rule describes a relevant part of the system's input/output behavior.

The PNC 2 cluster algorithm is initialized by treating each learn data tuple as a single cluster. Then, if a merge test is passed, iteratively always those two clusters with the same output value are merged, that are closest to each other. The merge test transforms the generalized cluster into a rule and evaluates it by a kind of hitrate. The rule's premise is the cuboid, that encloses the input vectors of all learn data tuples merged within the cluster. This representation suffers in high dimensional input spaces due to the COD problem and thus a special mechanism is used to extend the cuboid during the merge test. A heterogenous normalized and weighted Minkowski overlap metric is used to be able to process mixed continuous and nominal inputs. An integrated bagging component can improve accuracy and also reduces the time complexity for a learn data sample with N data tuples from $O(N^3)$ to approximately $O(N^2)$. The size of the learned rule set can be further reduced by applying a context sensitive feature selection, that individually removes the unnecessary inputs from each rule's premise. The algorithm can also be viewed as an instance based learning algorithm, namely as an exemplar-based generalization approach. Thus the idea of the k-nearest-neighbor algorithm (kNN), to base the decision on several surrounding learn data tuples, was transferred to improve the prediction accuracy.

The PNC 2 is compared experimentally with the most similar existing algorithms, namely with the NGE, the RISE and, of course, with the kNN algorithm. The PNC 2 outperforms the NGE algorithm and its variants and reaches better or comparative accuracies as the kNN or the RISE algorithm – with typically much smaller model sizes.

When doing experimentell comparisons, the free parameters of an algorithm must be tuned systematically – otherwise the results can be corrupted. Based upon a work of SALZBERG, the *tough* validation is defined and accomplished for the PNC 2 cluster algorithm. When doing tough validation, then the free parameters are tuned using cross-validation or a similar approach within the learn data sample. Previously, a preliminary study is executed with some development benchmarks to reduce the number of free parameters.

5.2 Outlook

Global Objectives and Evolutionary Optimization It is often possible to improve a learning algorithm's prediction accuracy by applying a feature selection or some optimization, that fine tunes the model in a post-processing step. Thereby global objectives and evolutionary algorithms can be used. Right now, no such approaches are employed within the PNC 2 cluster algorithm. Thus it should be investigated in future studies, if and how it is possible to further improve prediction accuracy or reduce model sizes by combining the PNC 2 cluster algorithm with feature selection methods and post-optimization to reduce the generated rule set with respect to some global decision criterions. Therefor, as the PNC 2 cluster algorithm has some similarities with IBL algorithms, it should be possible to transfer some known approaches [WM00b], that are used for a nearest neighbor algorithm to reduce the number of learn data tuples stored.

TSK-Models Within this work only rules of the form (1.2), that suggest a fixed output value in the consequent, were considered. This is adequately for classification tasks but, however, for regression tasks it is often more favorable to use TSK rules of the form (1.10). It could be tried to use the PNC 2 cluster algorithm to find a suitable partition of the input space, that is then used to learn local regression models.

Large Learn Data Samples An other aspect concerns the enhancement of the PNC 2 cluster algorithm to be able to handle very large learn data samples. A possible approach is the usage of so called *sampling* techniques, that reduce the learn data sample in a pre-processing step. The simplest strategy is the so called *uniform random sampling*, that just chooses some random data tuples from the original learn data sample. However, this approach may suffer from the problem, that aspects of the system's input/output behavior, that are only represented by very few learn data tuples, may be lost. An approach, that circumvents this problem, is the *density biased sampling* [PF00], for which data tuples are preferably chosen in parts of the input space with a low probability density. Another approach, that is used in [Dom97] together with the RISE algorithm, is the so called *windowing*: A fixed number of data tuples, for example $2\sqrt{N}$, is randomly chosen from the original learn data sample such that the possible output classes are evenly distributed. These data tuples build the data sample $\mathcal{P}_L$, whereas the remaining data tuples build the data sample $\mathcal{P}_{Rest}$. The data sample $\mathcal{P}_L$ is used to learn a model and those data tuples from the data sample $\mathcal{P}_{Rest}$, that are misclassified by this model, are then added to the data sample $\mathcal{P}_L$. Again a model is learned an so on. The whole procedure is repeated until an abortion criterion is met.

Alternatively it could be tried to transform the PNC 2 cluster algorithm into an incremental version. Thereby it is also possible to additionally use the sampling techniques described above. A first idea of how to realize an incremental version of the PNC 2 is drafted below: A basic model is learned with the original PNC 2 cluster algorithm using a manageable number of learn data tuples. Then, incrementally one after the other, the remaining learn data tuples are processed. Therefor a remaining data tuple is taken and transformed into an elementary cluster that is inserted into the cluster population. Then it is tried to merge this new cluster with one of the existing other clusters. There should be a threshold for the number of clusters and always the cluster with the lowest mass is deleted, if this threshold gets exceeded. The learn data tuples considered within the merge test build the so called *set of potentially negative examples*. This set is initialized with the data tuples used to learn the basic model. While working incrementally, always the oldest data tuple in this set is replaced by the actually processed one. Therewith this set is permanently updated. Moreover, adaptivity of the incremental version could be reached by shrinking the clusters' cuboid a little bit in each step.

COD Problem The COD problem affects the PNC 2 in such a way, that the clusters' cuboids tend to be smaller and smaller with an increasing number if inputs. Thus the anti COD mechanism according to eqn. (2.17) is used to extend the cuboids during the merge test. Some more work should be spent to further improve this mechanism or to try other alternative approaches to cope with the COD problem.

Order of the Merge Tests Another aspect is the order, in which it is tried to merge the clusters. Many unsupervised agglomerative cluster algorithms[1] use similarity criterions, that somehow try to consider the density or distance of the data tuples, that are merged within two clusters. Thus these approaches aim to get the latent structure of the data tuples in the input space. The strategy, used within the PNC 2, to chose always the pair of two clusters, whose cuboids have the lowest distance to each other, is quite similar to the approach of the *single linkage* algorithm. It is known, that this algorithm tends to produce clusters, that contain elongated chains of data tuples. The *complete linkage* algorithm, on the contrary, produces more compact clusters. Thus it should be tried to transfer the strategy of the complete linkage algorithm to the PNC 2. Also other strategies from other agglomerative cluster algorithms should be investigated.

[1] See section 1.7.2 for further information.

Appendix A

Overview of Benchmarks

The following enumeration gives a quick introduction to the benchmarks used. Table A.1 summarizes the most important characteristics of each benchmark like the number of different output classes S_y, the number of data tuples N or the number of inputs m. The abbreviations used to encode the origins of the benchmarks are itemized in table A.2.

Classification Tasks

- **Australian** Verification of credit card transactions based on anonymous customer information. The original data contained about 5% of data tuples with one or more missing feature values. These were substituted by the mean for continuous and by the most frequent symbol for nominal features.
- **Chess-End-Game** Decide if player *white* is able to win the game based upon a given situation on a chess board.
- **Cleveland** Decide if a patient suffers from a heart disease based upon the patient's record. The original output took integer values from 0 to 4, whereas 0 means, that the patient is healthy and 1 to 4 indicates, that the patient suffers with increasing degree from a heart disease.
- **DNA** Classify splice junctions in DNA sequences. Decide upon a given window of 60 nucleoid sequences, if there is an intron-exon, and exon-intron or no boundary in the middle of the window.
- **Hepatitis** Decide if a patient suffers from hepatitis based upon the patient's record.
- **House-Votes-84** Classify the party of a congress member based upon his votes to a couple of questions.
- **Iris** Classify iris flowers based upon sepal and petal length and width.
- **Letter-Recognition** Classify letters displayed in 20 different fonts based on some calculated features.
- **Mushrooms** Classify if a mushroom is definitely edible, definitely poisonous, or of unknown edibility. The latter case counts as poisonous.
- **Segmentation** Classify the type of surface for a pixel in a satellite image.
- **Waveform** Classify the two out of three possible waveforms, that were used to build a measured signal.
- **Waveform+Noise** The same as *Waveform*, but there are 19 additional inputs which contain nothing but random white noise.
- **Wine** Classify a wine's cultivar based on a chemical analysis.

Regression Tasks

- **Heat Exchanger** Prediction of the temperature of a heat exchanger in an industrial semi-batch reactor.

Table A.1: Benchmark Overview.

Benchmark	**Abbreviation**	S_y	N	m	**nom**	**cont**	**Missing values**	**Origin**
Australian	AUSTRALIAN	2	690	14	8	6	No	UCI
Cleveland	CLEVELAND	2	303	13	9	4	Yes	UCI
Chess-End-Game	CHESS	2	3196	36	36	0	No	UCI
DNA[1]	DNA	3	3175	60	0	60	No	Delve
Hepatitis	HEPATITIS	2	155	19	13	6	Yes	UCI
House-Votes-84	VOTES	2	435	16	16	0	Yes	UCI
Iris	IRIS	3	150	4	0	4	No	UCI
Letter-Recognition	LETTER	26	20000	16	0	16	No	UCI
Mushrooms	MUSHROOMS	2	8124	22	22	0	Yes	UCI
Segmentation	SEG	7	2130	19	0	19	No	UCI
Waveform	WAVE	3	5000	21	0	21	No	UCI
Waveform+Noise	WAVE+NOISE	3	5000	40	0	40	No	UCI
Wine	WINE	3	178	13	0	13	No	UCI
Heat Exchanger	TEMP	–	1126	3	0	3	No	ESR

Note: *nom* and *cont* indicates the number of nominal and continuous inputs

[1] The data sample used by Delve was build from the one of the UCI repository by removing 15 data tuples.

Table A.2: Origins.

Abbreviation	**Meaning**
Delve	Collection of benchmarks and definition of how to make comparable experiments – University of Toronto [RNH+96]. Delve is the abbreviation of *Data for Evaluation of Learning in Valid Experiments* (http://www.cs.utoronto.ca/~delve/).
ESR	Private benchmark used within the chair of electrical control engineering at the university of Dortmund.
UCI	*UCI Machine Learning Repository*: Collection of benchmarks – University of California, Irvine [BM98].

Appendix B

Results of the Experiments ExpA

The tables B.2 up to B contain the results from the experiments ExpA that are described in section 4.3. Table B.1 explains the abbreviations used.

Table B.1: Explanation of the abbreviations used in the tables B.2 up to B.

Abbreviation	**Explanation**
$\widehat{Q}_T$	Prediction accuracy on test data; MCE for classification and MAE for regression tasks
$\widehat{Q}_L$	Prediction accuracy on learn data; see above
K	Number of clusters
K_{Red}	Number of clusters after reduction, i.e. number of clusters whose mass exceeds the threshold p_{min}
ø m	Average number of inputs used in each cluster/rule – estimated over all non reduced clusters
Covered	Proportion of test data tuples whose input vector lies inside the cuboid of at least one cluster
ø HR	Hitrate averaged over all clusters
P_{H0}	Error probability using a two sided t-test for the case of paired samples, i.e. probability of the null hypothesis, that the observed difference of the prediction accuracy – compared to the default variant – is just perchance. The t-test is described in detail in appendix C.
Comp.	Difference of the prediction accuracy compared to the default variant in percent

Table B.2: Results for the benchmark Australian.

Alternative	$\widehat{Q}_T$	$\widehat{Q}_L$	K	K_{Red}	ø m	Covered	ø HR	P_{H0}	Comp.
Default	**14.61 %**	14.21 %	88.2	7.1	4.2	72.9 %	–	–	–
	± 1.41 %	± 1.40 %	± 20.4	± 5.7	± 1.2	± 12.8 %	–		
$\delta_{symb} = 1$	14.75 %	14.29 %	88	6.4	4.0	74.2 %	–	0.136	- 1.0 %
	± 1.56 %	± 1.77 %	± 22.5	± 5.7	± 1.2	± 13.0 %	–		
4σ-normalization	14.62 %	14.18 %	88.3	7.3	4.3	73.4 %	–	0.324	- 0.1 %
	± 1.39 %	± 1.40 %	± 21.6	± 5.7	± 1.1	± 12.7 %	–		
d_{avr}-normalization	14.64 %	14.12 %	93.0	8.8	4.5	69.2 %	–	0.085	- 0.2 %
	± 1.38 %	± 1.36 %	± 24.8	± 7.7	± 1.3	± 17.3 %	–		
Single	**14.53 %**	14.44 %	44.0	2.7	2.1	100.0 %	83.9 %	0.029	0.5 %
merge test	± 1.44 %	± 1.43 %	± 4.3	± 0.7	± 1.1	± 0.3 %	± 3.2 %		
MTB	**14.61 %**	14.21 %	88.2	7.1	4.2	72.9 %	–	1.000	0.0 %
	± 1.41 %	± 1.40 %	± 20.4	± 5.7	± 1.2	± 12.8 %	–		
Laplace corrected	**14.60 %**	14.20 %	88.2	7.1	4.2	72.9 %	85.4 %	0.208	0.0 %
hitrate	± 1.41 %	± 1.40 %	± 20.4	± 5.7	± 1.2	± 12.8 %	± 3.0 %		
Hitrate	**14.61 %**	14.16 %	88.2	7.1	4.2	72.9 %	94.6 %	0.283	0.0 %
	± 1.43 %	± 1.39 %	± 20.4	± 5.7	± 1.2	± 12.8 %	± 2.3 %		
Equidistant	**14.59 %**	14.28 %	115.0	7.7	4.2	64.2 %	–	0.341	0.1 %
discretization	± 1.43 %	± 1.49 %	± 40.0	± 6.4	± 1.8	± 17.6 %	–		
No feature	15.02 %	13.94 %	94.1	9.6	4.7	68.6 %	–	0.003	- 2.8 %
weights	± 1.84 %	± 1.58 %	± 25.7	± 7.6	± 1.1	± 17.6 %	–		
No CSFS	**14.53 %**	14.22 %	88.2	7.1	14.0	67.0 %	–	0.001	0.5 %
	± 1.40 %	± 1.40 %	± 20.4	± 5.7	± 0.0	± 12.6 %	–		

Table B.3: Results for the benchmark HEPATITIS.

Alternative	$\widehat{Q}_T$	$\widehat{Q}_L$	K	K_{Red}	ø m	Covered	ø HR	P_{H0}	Comp.
Default	**18.40 %**	9.41 %	12.1	5.7	4.3	67.9 %	–		
	± 4.08 %	± 3.29 %	± 2.4	± 1.7	± 0.8	± 11.9 %	–		
$\delta_{symb} = 1$	19.35 %	9.93 %	12.5	5.8	4.4	65.8 %	–	0.006	- 5.1 %
	± 4.04 %	± 3.00 %	± 2.9	± 1.8	± 0.8	± 12.3 %	–		
4σ-normalization	18.97 %	9.06 %	12.5	6.1	4.2	65.6 %	–	0.045	- 3.1 %
	± 4.07 %	± 3.25 %	± 2.6	± 1.7	± 0.7	± 12.0 %	–		
d_{avr}-normalization	19.16 %	9.15 %	12.5	6.1	4.5	66.0 %	–	0.022	- 4.1 %
	± 4.03 %	± 2.96 %	± 2.8	± 1.7	± 0.8	± 12.3 %	–		
Single	20.33 %	20.23 %	5.7	3.1	3.5	99.4 %	76.5 %	0.000	- 10.5 %
merge test	± 3.64 %	± 3.58 %	± 1.1	± 0.7	± 0.9	± 2.3 %	± 5.0 %		
MTB	18.44 %	9.51 %	12.1	5.7	4.3	67.9 %	–	0.160	- 0.2 %
	± 4.13 %	± 3.29 %	± 2.4	± 1.7	± 0.8	± 11.9 %	–		
Laplace corrected	**18.31 %**	8.89 %	12.1	5.7	4.3	67.9 %	83.8 %	0.035	0.5 %
hitrate	± 4.13 %	± 2.97 %	± 2.4	± 1.7	± 0.8	± 11.9 %	± 1.6 %		
Hitrate	18.45 %	7.52 %	12.1	5.7	4.3	67.9 %	96.3 %	0.227	- 0.3 %
	± 4.22 %	± 3.05 %	± 2.4	± 1.7	± 0.8	± 11.9 %	± 2.5 %		
Equidistant	**18.23 %**	8.99 %	12.7	5.9	4.2	66.5 %	–	0.322	0.9 %
discretization	± 4.59 %	± 3.27 %	± 2.6	± 1.7	± 0.8	± 12.1 %	–		
No feature	18.92 %	8.50 %	13.2	6.7	4.5	61.1 %	–	0.091	- 2.8 %
weights	± 4.26 %	± 3.18 %	± 2.7	± 2.0	± 0.8	± 11.9 %	–		
No CSFS	**17.89 %**	9.39 %	12.1	5.7	19.0	56.3 %	–	0.046	2.8 %
	± 4.12 %	± 3.01 %	± 2.4	± 1.7	± 0.0	± 12.7 %	–		

Table B.4: Results for the benchmark MUSHROOMS.

Alternative	$\widehat{Q}_T$	$\widehat{Q}_L$	K	K_{Red}	ø m	Covered	ø HR	P_{H0}	Comp.
Default	**0.67 %**	0.21 %	4.3	3.4	3.3	99.6 %	–		
	± 0.50 %	± 0.20 %	± 0.6	± 0.6	± 1.1	± 0.5 %	–		
4σ-normalization	**0.67 %**	0.21 %	4.3	3.4	3.3	99.6 %	–	1.000	0.0 %
	± 0.50 %	± 0.20 %	± 0.6	± 0.6	± 1.1	± 0.5 %	–		
d_{avr}-normalization	0.73 %	0.25 %	4.3	3.3	3.2	99.7 %	–	0.008	- 9.6 %
	± 0.53 %	± 0.22 %	± 0.7	± 0.6	± 1.1	± 0.5 %	–		
Single	3.82 %	3.27 %	3.2	3.0	3.7	99.8 %	93.4 %	0.000	- 470.7 %
merge test	± 2.10 %	± 2.00 %	± 0.4	± 0.3	± 0.8	± 0.4 %	± 2.8 %		
MTB	0.93 %	0.62 %	4.3	3.4	3.3	99.6 %	–	0.000	- 39.1 %
	± 0.54 %	± 0.44 %	± 0.6	± 0.6	± 1.1	± 0.5 %	–		
Laplace corrected	0.93 %	0.59 %	4.3	3.4	3.3	99.6 %	95.5 %	0.000	- 38.5 %
hitrate	± 0.56 %	± 0.41 %	± 0.6	± 0.6	± 1.1	± 0.5 %	± 2.3 %		
Hitrate	0.68 %	0.22 %	4.3	3.4	3.3	99.6 %	99.3 %	0.160	- 1.2 %
	± 0.51 %	± 0.21 %	± 0.6	± 0.6	± 1.1	± 0.5 %	± 0.6 %		
Equidistant	**0.67 %**	0.21 %	4.3	3.4	3.3	99.6 %	–	1.000	0.0 %
discretization	± 0.50 %	± 0.20 %	± 0.6	± 0.6	± 1.1	± 0.5 %	–		
No feature	0.75 %	0.23 %	4.3	3.3	3.3	99.6 %	–	0.008	- 11.9 %
weights	± 0.53 %	± 0.19 %	± 0.7	± 0.7	± 1.2	± 0.5 %	–		
No CSFS	**0.65 %**	0.21 %	4.3	3.4	22.0	98.3 %	–	0.239	3.3 %
	± 0.53 %	± 0.21 %	± 0.6	± 0.6	± 0.0	± 0.8 %	–		

Table B.5: Results for the benchmark SEG.

Alternative	$\widehat{Q}_T$	$\widehat{Q}_L$	K	K_{Red}	ø m	Covered	ø HR	P_{H0}	Comp.
Default	**4.82 %**	1.80 %	84.8	40.2	4.8	76.8 %	–		
	± 0.77 %	± 0.53 %	± 7.1	± 3.4	± 0.2	± 1.5 %	–		
4σ-normalization	5.02 %	1.90 %	84.8	40.1	4.8	77.0 %	–	0.041	- 4.2 %
	± 0.78 %	± 0.46 %	± 6.9	± 3.9	± 0.2	± 1.6 %	–		
d_{avr}-normalization	5.06 %	1.84 %	85.3	41.5	4.8	77.1 %	–	0.050	- 5.0 %
	± 0.72 %	± 0.60 %	± 6.2	± 3.2	± 0.2	± 1.2 %	–		
Single	5.44 %	2.47 %	52.2	24.6	5.3	86.0 %	78.4 %	0.000	- 13.0 %
merge test	± 0.78 %	± 0.47 %	± 10.3	± 2.1	± 0.2	± 1.7 %	± 1.7 %		
MTB	4.83 %	1.92 %	84.8	40.2	4.8	76.8 %	–	0.280	- 0.2 %
	± 0.77 %	± 0.53 %	± 7.1	± 3.4	± 0.2	± 1.5 %	–		
Laplace corrected	4.90 %	1.88 %	84.8	40.2	4.8	76.8 %	63.8 %	0.091	- 1.7 %
hitrate	± 0.82 %	± 0.51 %	± 7.1	± 3.4	± 0.2	± 1.5 %	± 1.8 %		
Hitrate	**4.74 %**	1.62 %	84.8	40.2	4.8	76.8 %	95.6 %	0.016	1.6 %
	± 0.75 %	± 0.51 %	± 7.1	± 3.4	± 0.2	± 1.5 %	± 0.8 %		
Equidistant	5.21 %	2.08 %	82.3	36.3	3.8	81.1 %	–	0.010	- 8.1 %
discretization	± 0.86 %	± 0.61 %	± 7.5	± 3.4	± 0.2	± 1.8 %	–		
No feature	4.94 %	1.70 %	88.5	43.6	5.0	76.2 %	–	0.244	- 2.6 %
weights	± 0.69 %	± 0.32 %	± 6.9	± 2.8	± 0.2	± 1.5 %	–		
No CSFS	**4.76 %**	1.74 %	84.8	40.2	19.0	65.6 %	–	0.324	1.2 %
	± 0.97 %	± 0.47 %	± 7.1	± 3.4	± 0.0	± 1.8 %	–		

Table B.6: Results for the benchmark TEMP.

Alternative	$\widehat{Q}_T$	$\widehat{Q}_L$	K	K_{Red}	ø m	Covered	ø HR	P_{H0}	Comp.
Default	**1.436**	1.048	55.3	27.6	1.4	86.3 %	–		
	± 0.381	± 0.277	± 6.5	± 4.4	± 0.1	± 1.8 %	–		
No output	2.177	1.833	55.3	27.6	1.4	86.3 %	–	0.000	- 51.7 %
recalculation c	± 0.436	± 0.393	± 6.5	± 4.4	± 0.1	± 1.8 %	–		
4σ-normalization	**1.421**	1.043	55.3	27.5	1.4	86.3 %	–	0.002	1.0 %
	± 0.376	± 0.271	± 6.4	± 4.4	± 0.1	± 1.9 %	–		
d_{avr}-normalization	**1.386**	1.055	55.4	27.5	1.4	86.2 %	–	0.002	3.5 %
	± 0.383	± 0.276	± 6.4	± 4.4	± 0.1	± 1.9 %	–		
Single	1.943	1.555	37.9	22.5	1.4	93.5 %	87.1 %	0.002	- 35.3 %
merge test	± 0.912	± 0.779	± 3.3	± 3.1	± 0.1	± 2.4 %	± 2.0 %		
MTB	**1.436**	1.048	55.3	27.6	1.4	86.3 %	–	1.000	0.0 %
	± 0.381	± 0.277	± 6.5	± 4.4	± 0.1	± 1.8 %	–		
Laplace corrected	1.464	1.075	55.3	27.6	1.4	86.3 %	37.3 %	0.000	- 2.0 %
hitrate	± 0.390	± 0.274	± 6.5	± 4.4	± 0.1	± 1.8 %	± 2.5 %		
Hitrate	1.437	1.047	55.3	27.6	1.4	86.3 %	97.5 %	0.121	- 0.1 %
	± 0.380	± 0.276	± 6.5	± 4.4	± 0.1	± 1.8 %	± 0.8 %		
Equidistant	**1.434**	1.048	55.3	27.5	1.4	86.2 %	–	0.343	0.1 %
discretization	± 0.373	± 0.274	± 6.5	± 4.5	± 0.1	± 1.8 %	–		
No feature	1.440	1.037	54.7	27.6	1.4	86.5 %	–	0.333	- 0.3 %
weights	± 0.363	± 0.260	± 6.2	± 4.1	± 0.1	± 1.8 %	–		
No CSFS	**1.414**	1.074	55.3	27.6	3.0	82.7 %	–	0.153	1.5 %
	± 0.367	± 0.283	± 6.5	± 4.4	± 0.0	± 2.4 %	–		

Appendix C

t-Test for Paired Samples

Student's t-test for the case of paired samples [PTVF93] (P. 618)[1] is used to decide, if the mean of two samples Q_a and Q_b is significantly different. The two samples must be somehow paired to each other, e.g. measured values of two different sensors for the same object. Here Q_a and Q_b is the prediction accuracy of two different models evaluated for N_R different learn and test sets. The t statistic is calculated as

$$t = \frac{\overline{Q}_a - \overline{Q}_b}{s} \tag{C.1}$$

with

$$s = \sqrt{\frac{\sigma_{Q_a}^2 + \sigma_{Q_b}^2 - 2\,Cov(Q_a, Q_b)}{N_R}} \tag{C.2}$$

and

$$Cov(Q_a, Q_b) = \frac{1}{N_R - 1} \sum_{w=1}^{N_R} (Q_{a_w} - \overline{Q}_a)(Q_{b_w} - \overline{Q}_b) \ , \tag{C.3}$$

whereas $\overline{Q}_a$ resp. $\overline{Q}_b$ denotes the mean and $\sigma_{Q_a}^2$ resp. $\sigma_{Q_b}^2$ the variance of the particular variable. The probability of the null hypothesis P_{H0} is determined from the value of the t statistic using student's two sided t-distribution with a degree of freedom of $\nu = N_R - 1$ as

$$P_{H0} = I_{\frac{\nu}{\nu + t^2}} \left(\frac{\nu}{2}, 0.5 \right) \tag{C.4}$$

with the so called *incomplete beta function* [PTVF93] (P. 226ff.)

$$I_x(a, b) = \frac{1}{B(a, b)} \int_0^x t^{a-1} (1 - t)^{b-1} dt \tag{C.5}$$

and the so called *beta function* [PTVF93] (P. 215f.)

$$B(a, b) = \int_0^1 t^{b-1} (1 - t)^{a-1} dt \ . \tag{C.6}$$

[1] The referenced book contains *C* source code of the test and of all functions additionally needed.

Appendix D

Symbols, Abbreviations and Indices

Table D.1: Abbreviations.

Abbreviation	**Meaning**
COD	Curse of Dimensionality
COG	Center of Gravity
BNGE	Batch NGE *algorithm*
FCM	Fuzzy C-Means *algorithm*
GK	Gustafson-Kessel *algorithm*
GG	Gath-Geva *algorithm*
HEOM	Heterogeneous Euclidean Overlap Metric
HR	Hitrate
HVDM	Heterogeneous VDM
IBL	Instance Based Learning
IVDM	Interpolated VDM
*k*NN	*k* Nearest-Neighbor *algorithm*
CSFS	Context Sensitive Feature Selection
LOOCV	Leave-One-Out Cross-Validation
MAE	Mean Absolute Error
MCE	Mean Classification Error
MFC	Most Frequent Class
MND	Mutual Neighbor Distance
MOM	Mean of Maximum *defuzzification*
MSF	Membership Function
MTB	Mass Tie Breaking *policy*
MDLP	Minimum Description Length Principle
MSE	Mean Square Error
MMC	Mean Misclassification Costs
NGE	Nearest Generalized Exemplar *algorithm*
NN	Nearest-Neighbor *algorithm*
OBNGE	Overlapping Batch NGE *algorithm*
PNC 2	Positive and Negative Example Based Clustering
RISE	Rule Induction from a Set of Exemplars *algorithm*
TSK	Takagi-Sugeno-Kang *model*
VDM	Value Difference Metric

Table D.2: Special Terms.

Term	Meaning
Data sample	Sample with data tuples.
Data tuple	In the context of data-based modelling a data tuple consists of an input vector and an associated output value, whereas in the context of unsupervised cluster algorithms a data tuple is just a data vector.
Input vector	Vector with m inputs. The term *input position* is preferred when using a model without any relation to a data sample.
Data vector	Vector with m variables.

Table D.3: Special Counters and Indices.

Counter	Index	Kontext	Example
N	i	Data tuples	Data tuple P_i
m	j	Inputs/variables	Input/variable x_j
K	t	Cluster	Cluster C_t
S_j, S_x	w, z	Symbols of nominal or discretized inputs	Symbol w bzw. z
S_y	s	Symbols of nominal or discretized output	Symbol s
–	T, L	Test and learn data sample	Prediction accuracy Q_T and Q_L
–	a, b	Two elements	Data tuples P_a and P_b
–	w, z	Common indices used for matrices etc.	

Table D.4: Globally used Symbols.

Symbol	Meaning
$\mathbf{b}$	Bit string that encodes the symbols, that are covered by a cuboid with respect to a nominal input
C	Cluster
c	Cluster output value or conclusion
$\mathbf{B}$	Misclassification cost matrix with the elements $b_{w,z}$
e	Prediction error
$\mathcal{C}$	Cluster population
d	Distance of two clusters/data tuples
d_{avr}	Average component wise distance with respect to an input x_j
d_j	Component wise distance with respect to an input x_j
E	Exemplar (NGE algorithm)
$E(y)$	Entropy of the output
$f_{tf}(\mathbf{x})$	Underlying true function
H	Cuboid
$I(x, y)$	Transinformation of an input to the output
l	Lower left bound of a cuboid with respect to a continuous input
k	Parameter of the kNN algorithm
K_{Red}	Number of clusters whose mass exceeds the threshold p_{min}
N_{Bins}	Number of discretization intervals (continuous inputs)
N_R	Number of repetitions or cross-validations for estimation of the prediction accuracy Q
N_{Tune}	Number of repetitions or cross-validations to tune parameters
$P(\mathbf{x}, y)$	Data tuple
P_{H0}	Probability of the null hypothesis
$P(c\|p)$, $\widehat{P}(c\|p)$	Conditional probability of the conclusion given the premise
p	Mass of a cluster/number of positive examples or conclusion
$\mathcal{P}$	Data sample with N data tuples
n	Number of negative examples
$\mathbf{O}$	Confusion matrix with the elements $o_{w,z}$
Q^*, $\widehat{Q}$	Real and estimated accuracy of a learning algorithm
Q_P^*, $\widehat{Q}_P$	Real and estimated accuracy of a model
r	Upper right bound of a cuboid with respect to a continuous input

W_{Kernel}	Parameter PNC 2 cluster algorithm
$W_{Kernel\,Min}$	Parameter PNC 2 cluster algorithm
$\mathbf{x}$	Input vector or data vector
$\overline{x}$	Mean of an input
x_{max}	Maximum of an input
x_{min}	Minimum of an input
x_{range}	Range of an input
y	Output value
$\widehat{y}$	Predicted output value
α	Error probability
δ	Normalization factor
η	Parameter PNC 2 cluster algorithm
$\mu(\mathbf{x})$	Truth value of a fuzzy rule's premise
$\mu(y)$	Truth value of a fuzzy rule's consequent
$\mu_{tf}(\mathbf{x})$	Distribution of the input vectors $\mathbf{b}$
ν	Degree of freedom
ξ	Degree of coverage of a cuboid's component
ρ	Parameter Minkowski metric
ϱ	Hitrate of a cluster/rule
σ_Q	Standard deviation prediction accuracy
σ'_Q	Standard error of estimated prediction accuray
σ_j	Standard deviation of input x_j
σ_y	Standard deviation of the output
σ_{MSF}	Parameter PNC 2 cluster algorithm
ς	Parameter PNC 2 cluster algorithm
ω_{COD}	Parameter PNC 2 cluster algorithm
ω	Feature weight

Bibliography

[Aha90] David W. Aha. A study of instance-based learning algorithms for supervised learning tasks: Mathematical, empirical, and psychological evaluations, 1990.

[Aha95a] David Aha. An implementation and experiment with the nested generalized exemplars algorithm. Technical Report AIC–95–003, Naval Research Laboratory, 1995.

[Aha95b] David W. Aha. Machine learning. In *5th International Workshop on Artificial Intelligence & Statistics*, Ft. Lauderdale, FL, January 1995. Tutorial slides.

[Aha98] David W. Aha. Feature weighting for lazy learning algorithms, 1998.

[AHK01] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. *Lecture Notes in Computer Science*, 1973, 2001.

[And73] M. R. Anderberg. *Cluster analysis for applications*. Academic Press Inc., New York, 1973.

[Bab98] Robert Babuska. *Fuzzy modeling for control*. Kluwer Academic Publisher, Boston, 1998.

[Bab02] Robert Babuska. Fuzzy systems, modeling and identification, 2002. Lecture Notes, Delft University of Technology, Department of Electrical Engineering, The Netherlands.

[Bel61] R. E. Bellman. *Adaptive control processes: A guided tour*. Princeton University Press, Princeton, 1961.

[Bez81] J.C. Bezdek. *Pattern recognition with fuzzy objective function algorithms*. Plenum Press, New York, USA, 1981.

[BGC97] Ben Burdsall and Christophe Giraud-Carrier. Evolving fuzzy prototypes for efficient data clustering. In *Proc. 2nd International ICSC Symposium on Fuzzy Logic and Applications (ISFL)*, pages 217–223. ICSC Academic Press, February 1997.

[BK99] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1–2):105–139, 1999.

[BM98] C.L. Blake and C.J. Merz. Uci repository of machine learning databases. Technical report, Department of Information and Computer Science, University of California, Irvine, 1998.

[CH67] T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. In *IEEE Transactions on Information Theory*, volume 13, pages 21–27, January 1967.

[Das91] Belur V. Dasarathy. Nearest neighbor NN norms: Nn pattern classification techniques. In *IEEE Computer Society Press*, pages 388–397, Los Alamitos, 1991.

[DH73] Richard Duda and Peter Hart. *Pattern recognition and scene analysis*. John Wiley and Sons, New York, 1973.

[Die98] Thomas G. Dietterich. Machine-learning research: Four current directions. *The AI Magazine*, 18(4):97–136, 1998.

[DKS95] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In *Proc. International Conference on Machine Learning*, pages 194–202, 1995.

[DM98] Kan Deng and Andrew Moore. On the greediness of feature selection algorithms. In *Proc. International Conference on Machine Learning (ICML)*, June 1998.

[Dom95] Pedro Domingos. Rule induction and instance-based learning: A unified approach. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1226–1232, 1995.

[Dom96] Pedro Domingos. Unifying instance-based and rule-based induction. In *Proc. International Conference on Machine Learning*, volume 24, pages 141–168, 1996.

[Dom97] Pedro Domingos. *A unified approach to concept learning.* PhD thesis, Department of Information and Computer Science, University of California, Irvine, 1997.

[Eve93] B. S. Everitt. *Cluster analysis.* Edward Arnold Ltd., London, UK, 1993.

[FBY92] William B. Frakes and Ricardo A. Baeza-Yates, editors. *Information retrieval: Data structures & algorithms.* Prentice-Hall, Upper Saddle River, NJ, 1992.

[FI89] Usama M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proc. 13th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 800–805, 1989.

[Fle96] Arthur Flexer. Statistical evaluation of neural network experiments: Minimum requirements and current practice. In *Proc. 13th European Meeting on Cybernetics and Systems Research*, volume 2, pages 1005–1008, A–1010 Vienna, Austria, 1996.

[Fri97] Bernd Fritzke. Incremental neuro-fuzzy systems. In *Applications of Soft Computing, SPIE International Conference*, 1997.

[GG89] I. Gath and A.B. Geva. Unsupervised optimal fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 773–781, July 1989.

[GK79] D. Gustafson and W. Kessel. Fuzzy clustering with a fuzzy covariance matrix. *Proc. IEEE CDC*, pages 761–766, 1979.

[Goo65] I. J. Good. *The estimation of probabilities: An essay on modern bayesian methods.* MIT Press, Cambridge, 1965.

[GSJ97] A.F. Gómez-Skarmeta and F. Jiménez. Generating and tuning fuzzy rules using hybrid systems. In *Proc. 6th International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 247–252, Barcelona, Spain, 1997.

[Hae01a] Lars Haendel. Data-based lerning of fuzzy-rules using an agglomerative cluster algorithm. In *Proc. European Symposium on Intelligent Technologies, Hybrid Systems and their implementation on Smart Adaptive Systems (Eunite)*, Puerto de la Cruz, Tenerife, Spain, December 2001.

[Hae01b] Lars Haendel. Datenbasierte generierung von fuzzy-regeln mittels eines agglomerativen clusterverfahrens. In *Proc. 11. Workshop Fuzzy-Control des GMA-FA 5.22*, October 2001.

[HAK00] Alexander Hinneburg, Charu C. Aggarwal, and Daniel A. Keim. What is the nearest neighbor in high dimensional spaces? *The VLDB Journal*, pages 506–515, 2000.

[HC99] Nicholas Howe and Claire Cardie. Weighting unusual feature types. Technical Report TR99–1735, Cornell University, 9, 1999.

[HK00] Frank Höppner and Frank Klawonn. Obtaining interpretable fuzzy models from fuzzy clustering and fuzzy regression. In *Proc. KES00*, pages 162–165, Brighton, UK, 2000.

[HKK97] Frank Höppner, Frank Klawonn, and Rudolf Kruse. *Fuzzy-Clusteranalyse: Verfahren für die Bilderkennung, Klassifizierung und Datenanalyse.* Vieweg, Braunschweig; Wiesbaden, 1997.

[HLTM99] F. Hussain, H. Liu, C. L. Tan, and Dash M. Discretization: An enabling technique. Technical Report TRC6/99, School of Computing, National University of Singapore, 1999.

[IV94] Ibrahim Imam and Haleh Vafaie. An empirical comparison between global and greedy-like search for variable selection. In *Florida AI Research Symposium (FLAIRS)*, 1994.

[JKP94] George H. John, Ron Kohavi, and Karl Pfleger. Irrelevant features and the subset selection problem. In *Proc. International Conference on Machine Learning*, pages 121–129, 1994. Journal version in AIJ.

[JMF00] A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31, 2000.

[JP73] R.A. Jarvis and E.A. Patrick. Clustering using a similarity measure based on shared near neighbors. *IEEE Transaction on Computers*, C-22(11), November 1973.

[KHDM98] J. Kittler, M. Hatef, R. P.W. Duin, and J. Matas. On combining classifiers. In *Proc. IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 20, march 1998.

[KHJ97] Mark Kantrowitz, Erik Horstkotte, and Cliff Joslyn. Answers to frequently asked questions about fuzzy logic and fuzzy expert systems, 1997. Official FAQ of the Usenet newsgroup `comp.ai.fuzzy` (ftp.cs.cmu.edu:/user/ai/pubs/faqs/fuzzy/fuzzy.faq).

[Kie97] Harro Kiendl. *Fuzzy-Control methodenorientiert.* Oldenbourg Verlag, München, 1997.

[Kin67] B. King. Step-wise clustering procedures. *Journal of the American Statistical Association*, 69:86–101, 1967.

[KJ97] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1–2):273–324, 1997.

[KK97] Frank Klawonn and Annette Keller. Fuzzy clustering and fuzzy rules. In *Proc. 7th International Fuzzy Systems Association World Congress IFSA'97*, volume 1, pages 193–198. Academica Prague, 1997.

[Koh95] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1137–1145, 1995.

[KW00] Ivan Kojadinovic and Thomas Wottka. Comparision between a filter and a wrapper approach to variable subset selection in regression problems. In *Proc. European Symposium on Intelligent Techniques (ESIT)*, September 2000.

[LWTB97] W. Z. Liu, A. P. White, S. G. Thompson, and M. A. Bramer. Techniques for dealing with missing values in classification. *Lecture Notes in Computer Science*, 1280, 1997.

[McQ67] J. McQueen. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkely Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[MG81] E. H. Mamdani and B. R. Gaines. *Fuzzy reasoning and its applications.* Academic Press, London, 1981.

[Mur83] F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4), 1983.

[Nad64] E. A. Nadaraya. On estimating regression. *Theory of Probability and Its Application*, 10:186–190, 1964.

[Nag68] G. Nagy. State of the art in pattern recognition. In *Proceedings IEEE*, volume 56, pages 836–862, 1968.

[Nib87] T. Niblett. Constructing decision trees in noisy domains. In *Proc. 2nd European Working Session on Learning*, pages 67–78, Bled, Yogoslavia, 1987.

[NK97] Detlef Nauck and Rudolf Kruse. What are neuro-fuzzy classifiers? In *Proc. 7th International Fuzzy Systems Association World Congress IFSA'97*, volume 4, pages 228–233. Academica Prague, 1997.

[PF00] Christopher R. Palmer and Christos Faloutsos. Density biased sampling: an improved method for data mining and clustering. In *Proc. ACM International Conference on Management of Data (SIGMOD)*, pages 82–92, Dallas, Texas, United States, 2000.

[PTVF93] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C.* Cambridge University Press, Cambridge, UK, 1993. 2nd Edition.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Qui89] J. R. Quinlan. Unknown attribute values in induction. In *Proc. 6th International Machine Learning Workshop*, pages 164–168, 1989.

[Rez94] Fazlollah M. Reza. *An introduction to information theory.* New York, Dover, 1994.

[RNH+96] C. E. Rasmussen, R. M. Neal, G. E. Hinton, D. van Camp, M. Revow, Z. Ghahramani, R. Kustra, and R Tibshrani. The delve manual. Technical report, University of Toronto, 1996.

[RPP99] A. Dourado e B. Duarte R. P. Paiva. Applying subtractive clustering for neuro-fuzzy modelling of a bleaching plant. In *Proc. European Control Conference ECC*, 1999.

[RSA00] J. Roubos, M. Setnes, and J. Abonyi. Learning fuzzy classification rules from data. In *Proc. RASC Conference*, Leichester, UK, 2000.

[Sal91] Steven Salzberg. A nearest hyperrectangle learning method. In *Proc. 6th International Conference on Machine Learning*, pages 251–276, 1991.

[Sal97] Steven Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1(3):317–328, 1997.

[Sal99] Steven Salzberg. *On comparing classifiers: A critique of current research and methods.* Kluwer Academic Publishers, Boston, 1999.

[Sar02] Warren S. Sarle. Neural network faq, 2002. Official FAQ of the Usenet newsgroup `comp.ai.neural-nets` (ftp://ftp.sas.com/pub/neural/FAQ.html).

[SBBG02] Sergio M. Savaresi, Daniel L. Boley, Sergio Bittani, and Giovanni Gazzaniga. Cluster selection in divisive clustering algorithms. In *Proc. 2nd SIAM International Conference on Data Mining*, April 2002.

[Sha48] C.E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:379–423 and 623–656, 1948.

[Sla01] Timo Slawinski. *Analyse und effiziente Generierung von relevanten Fuzzy-Regeln in hochdimensionalen Suchräumen.* PhD thesis, Universität Dormund, Fakultät für Elektrotechnik, 2001. VDI Verlag Düsseldorf, Fortschrittberichte VDI, Reihe 10, Nr. 686.

[SS73] P. H. A. Sneath and R. R. Sokal. *Numerical taxonomy.* W. H. Freeman, San Francisco, 1973.

[SW92] Craig Stanfill and David L. Waltz. *Statistical methods, artificial intelligence, and information retrieval.* Lawrence Erlbaum Associates, 1992.

[TS85] T. Takagi and M. Sugeno. Fuzzy identification of systems and its application to modeling and control. *IEEE Transactions on Systems, Man, and Cybernetics*, 15(1):116–132, 1985.

[WAM97] Dietrich Wettschereck, David W. Aha, and Takao Mohri. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review*, 11(1–5):273–314, 1997.

[War63] J. H. Ward. Hierarchical grouping to optimize an objective function. *Journal of American Statistical Association*, 58:236–245, 1963.

[Wat64] G.S. Watson. Smooth regression analysis. *Sankhya: The Indian Journal of Statsitics Series A*, 26:359–372, 1964.

[WD95] Dietrich Wettschereck and Thomas G. Dietterich. An experimental comparison of the nearest-neighbor and nearest-hyperrectangle algorithms. In *Proc. International Conference on Machine Learning*, volume 19, pages 5–27, 1995.

[Wet94] Dietrich Wettschereck. A hybrid nearest-neighbor and nearest-hyperrectangle algorithm. In *Proc. European Conference on Machine Learning*, volume 784, pages 323–335, 1994.

[WM96] D. Randall Wilson and Tony R. Martinez. Value difference metrics for continuously valued attributes. In *Proc. Iternational Conference on Artificial Intelligence, Expert Systems and Neural Networks*, pages 11–14, 1996.

[WM97a] D. Randall Wilson and Tony R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997.

[WM97b] D. Randall Wilson and Tony R. Martinez. Instance pruning techniques. In *Proc. 14th International Conference on Machine Learning*, pages 403–411. Morgan Kaufmann, 1997.

[WM00a] D. Randall Wilson and Tony R. Martinez. An integrated instance-based learning algorithm. *Computational Intelligence*, 16(1):1–28, 2000.

[WM00b] D. Randall Wilson and Tony R. Martinez. Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38(3):257–286, 2000.

[Zad65] Lotfi A. Zadeh. Fuzzy sets. *Information Control*, 8:338–353, 1965.

[Zad68] Lotfi A. Zadeh. Probability measures of fuzzy events. *Journal of Mathematical Analysis and Applications*, 23:421–427, 1968.